Notes on Combinatorial Optimization

Robert E. Bixby *
October 16, 1987

TR87-21

*Partially supported by AFOSR grant 87-0276 to Rice University.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
16 OCT 1987		**00-00-1987 to 00-00-1987**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Notes on Combinatorial Optimization	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Computational and Applied Mathematics Department ,Rice University,6100 Main Street MS 134,Houston,TX,77005-1892**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**70**	

Contents

Chapter 1

Introduction

1.1 What is Combinatorial Optimization?

Graphs

Most combinatorial optimization problems are defined on graphs. We assume some familiarity with this subject and give only a short introduction. Most of the books referenced at the end of these notes also contain introductory sections on graphs.

An (*undirected*) *graph* G is a pair (E, V), where V is a finite set of *vertices* and E is a finite set of *edges*. Each edge has associated with it two vertices, not necessarily distinct, called its *ends*. An edge with identical ends is a *loop*. Two non-loop edges with the same ends are said to be *parallel*. If G has no loops or parallel edges it is *simple*.

It is typical to draw pictures of graphs in which vertices are depicted as points and edges as line segments joining these points. The graph pictured in Figure 1.1 has 4 vertices, 6 edges, one loop, and two parallel edges. There is one labeled edge, e, and there are two labeled vertices, x and y, the ends of e. When there is no ambiguity, as there is not for a simple graph, we write $e = xy$.

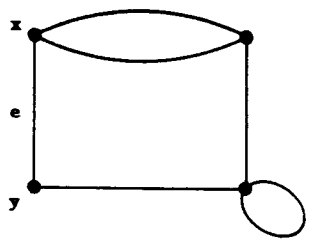


Figure 1.1: An example graph.

A *path* P from a vertex v_0 to a vertex v_k in a graph G, sometimes called a v_0–v_k *path*, is a sequence of vertices and edges of G, $P = (v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k)$, such that $e_i = v_{i-1}v_i$ $(i = 1, \ldots, k)$. When no ambiguity arises we write $P = (v_0, \ldots, v_k)$. If $v_0, \ldots, v_k$ are distinct, the path is called *simple*. A path that is simple except that $v_0 = v_k$ is called a *circuit*. The graph in Figure 1.1 contains exactly four distinct circuits. Note that a loop ia a circuit with exactly one edge (and one vertex). A pair of parallel edges also forms a circuit.

A graph is *connected* if for every pair of vertices x and y, there is a path from x to y. A graph is a *forest* if it includes no circuits. A connected forest is a *tree*. A *subgraph* of a given graph is obtained by deleting some of its vertices and edges. Of course, when a vertex is deleted from a graph, then all incident edges must also be deleted. An edge can be deleted without deleting its end-vertices. A *spanning subgraph* of a graph is one in which only edges have been deleted. A *spanning tree* is a spanning subgraph that is a tree. The graph in Figure 1.1 has exactly 7 spanning trees.

A Definition of Combinatorial Optimization

A combinatorial optimization problem can be defined in general as follows.

Definition 1.1.1 Let E be a finite set, let $\mathcal{S}$ be a family of subsets of E, and let $w \in \mathbf{R}^E$ be a real-valued weight function defined on the elements of E. The associated *combinatorial optimization* (CO) problem is to find $S^* \in \mathcal{S}$ such that

$$w(S^*) = \min_{S \in \mathcal{S}} w(S),$$

where $w(S) := \sum_{e \in S} w(e)$. □

Example 1.1.2 *Traveling Salesman Problem.* Let K_n denote the *complete graph* on n vertices. Thus, K_n is a simple graph on n vertices in which every two vertices are joined by an edge. Let w be a weight function defined on the edges of K_n. In the typical traveling-salesman interpretation the vertices of K_n represent cities, and the weights distances between these cities. Of course, other interpretations are also possible, and important.

The n-city traveling salesman problem is to construct a circuit that passes through each vertex, and has minimum total weight. A circuit passing through each vertex of a graph is called a *tour*. In the graph K_4 given in Figure 1.2, and for the weight function indicated on the edges, the minimum tour has weight 8.

To formulate the TSP as a CO problem, take S to be the set of edges of K_n (so that $|S| = n(n-1)/2$), and let $\mathcal{S}$ be the family of subsets of S forming tours. The TSP is an example of a hard CO problem. It is known to be $\mathcal{NP}$-complete: No polynomial-time algorithm is known for it, and none is expected. □

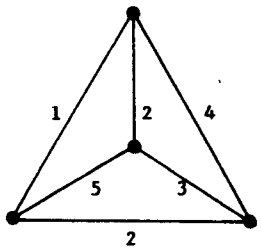


Figure 1.2: K_4

Example 1.1.3 *Minimum Spanning Trees.* Let G be a connected graph. Given a weight function defined on the edges of G, the minimum spanning tree problem is to find a spanning tree of G that has minimum total weight. This is clearly a CO problem.

The MST problem has numerous applications. One simple one is the following. Suppose a collection of remote computer installations has been specified, and a cost is given for connecting each separate pair of terminals by a direct communication link. It is reasonable to ask for a minimum-cost collection of links the construction of which would allow any one terminal to communicate with any other, communication being possible exactly when there is path of links between the two terminals in question. Given that the costs are nonnegative (and who would doubt that they are), this is exactly a MST problem.

The MST problem is an example of an easy CO problem. There are several polynomial-time algorithms known to solve it. □

1.2 Independence Systems

In this section we develop a general algorithm that can be applied to a wide variety of CO problems. For most, it is only a heuristic, but for the MST problem we will see that it gives an exact solution.

Definition 1.2.1 Let E be a finite set, and let $\mathcal{I}$ be a family of *independent subsets* of E. The pair $(E, \mathcal{I})$ is called an *independence system* if

(I1) the empty set is independent, and

(I2) subsets of independent sets are independent. □

Example 1.2.2 Both the TSP and the MST problem can be formulated as CO problems on independence systems. For example, in the MST problem one can

take as the underlying set E the set of edges of the given graph G, and as the family $\mathcal{I}$ the family of edge-sets of forests of G. The MST problem for G and a weight function w is then equivalent to finding a maximum-weight independent subset of E with respect to weight function $\{M - w(e) : e \in E\}$, for a sufficiently large constant M.

The TSP may be handled similarly. □

The following algorithm, named by Jack Edmonds, can be applied to any independence system.

Algorithm 1.2.3 The Greedy Algorithm.

Input: An independence system $(E,\mathcal{I})$, a weight function $w \in \mathbf{R}^E$, and an independence oracle.

Output: An independent set I_g (of hopefully large total weight).

Comment: In order to apply an algorithm to $(E,\mathcal{I})$, $\mathcal{I}$ must be specified in some computationally accessible form. The typically assumed form is that of an ***independence oracle.*** Thus, it is assumed that a subroutine or "oracle" is available that can determine in constant time if a given $X \subseteq E$ is independent. In applications this oracle is replaced by a concrete calculation.

begin
  sort E as $e_1, \ldots, e_{|E|}$ so that for some k
  $w(e_1) \geq \ldots \geq w(e_k) > 0 \geq w(e_{k+1}) \geq \ldots \geq w(e_{|E|})$;
  $I_g := \emptyset$;
  for $j := 1$ **until** k **do**
    if $I_g \cup \{e_j\} \in \mathcal{I}$ **then** $I_g := I_g \cup \{e_j\}$;
end

Note that this algorithm requires at most $|E|$ calls to the independence oracle; moreover, for any reasonable implementation, the time requirements in addition to these calls are bounded by a polynomial in $|E|$. The algorithm is thus said to run in *oracle polynomial time.* More detailed estimates for concrete instances will be given later.

For $X \subseteq E$, a *base* B of X is a maximal independent subset of X, where B *maximal* independent means that for every $e \in X \backslash B$, $B \cup \{e\}$ is not independent. The *rank* of X, $r(X)$, is defined by

$$r(X) = \max\{|B| : B \text{ a base of } X\}.$$

The *lower rank* of X, $r_l(X)$, is defined by

$$r_l(X) = \min\{|B| : B \text{ a base of } X\}.$$

In terms of these quantities, we have the following general performance bound on the greedy algorithm.

Theorem 1.2.4 (Jenkyns 1976) *Where I_o is an optimal-weight independent set and $w(I_o) > 0$,*

$$\min\{\frac{r_l(F)}{r(F)} : F \subseteq E, r(F) \neq 0\} \leq \frac{w(I_g)}{w(I_o)} \leq 1.$$

Proof. Let w_i denote w_{e_i} and write $E_i = \{e_1, \ldots, e_i\}$ $(i = 1, \ldots, k)$. Define $w_{k+1} = 0$. Then we have

$$w(I_g) = \sum_{j=1}^{k} |I_g \cap E_j|(w_j - w_{j+1}),$$

and

$$w(I_o) = \sum_{j=1}^{k} |I_o \cap E_j|(w_j - w_{j+1}).$$

But for each j, by the nature of the greedy algorithm, $I_g \cap E_j$ is a maximal independent subset of E_j. Hence, $r_l(E_j) \leq |I_g \cap E_j|$; moreover, $|I_o \cap E_j| \leq r(E_j)$ since $I_o \cap E_j$ is independent (being a subset of I_o) and $r(E_j)$ is the size of a biggest independent subset of E_j. Combining the above facts, and denoting the 'min' in the theorem by q, we have

$$\begin{aligned} w(I_g) &\geq \sum_{j=1}^{k} r_l(E_j)(w_j - w_{j+1}) \\ &\geq \sum_{j=1}^{k} q\, r(E_j)(w_j - w_{j+1}) \\ &\geq q \sum_{j=1}^{k} |I_o \cap E_j|(w_j - w_{j+1}) \\ &= q\, w(I_o), \end{aligned}$$

as required. □

Corollary 1.2.5 *If $r(F) = r_l(F)$ for all $F \subseteq E$, then the greedy algorithm gives an optimal solution of the maximum independent set problem.* □

The objects singled out by (1.2.5) are known in the literature as "matroids" (see [2]). We will not study their general properties further here.

Exercises

1.1 Let $\mathcal{I}$ be the family of subsets of edge-sets of tours in K_n. Define the rank r and lower rank r_l for $\mathcal{I}$ as in §1.2. Let

$$q = \min\{\frac{r_l(F)}{r(F)} : F \subseteq E(K_n) \text{ and } r(F) \neq 0\}.$$

Show that $q \geq 1/2$.

Chapter 2

Minimum Spanning Trees

2.1 The Greedy Algorithm

An interesting overview of the subject of minimum spanning trees is given in "On the history of the minimum spanning tree problem," by R. L. Graham and P. Hell (1985), *Annals of the History of Computing* **7** 43–57.

There are several known polynomial-time algorithms for the MST problem. We begin by showing that the greedy algorithm (Algorithm 1.2.3) is one of them.

We make use here and throughout the remainder of these notes of the word 'maximal' in a set-theoretic sense. Thus, an (edge) *maximal* forest is one such that the addition of any edge in the given graph, outside the forest, destroys the forest property (creates a circuit). The word 'minimal' is used similarly. Hence, there is an important difference between the words 'maximum' and 'minimum' and the words 'maximal' and 'minimal'. (See also the definition of 'base' in §1.2.)

A (*connected*) *component* of a graph G is a maximal connected subgraph.

Lemma 2.1.1 *(a) Every (edge) maximal forest of a connected graph is a spanning tree. (b) Every spanning tree of a connected graph on n vertices has exactly $n-1$ edges.*

Proof. Let G be a connected graph on n vertices. The lemma is trivial if $n = 1$. Assume $n > 1$.

We first prove (a). Let T be a maximal forest of G. If T is not spanning, let x be a vertex of G not incident to an edge of T. Since G is connected, and has at least two vertices, there must be an edge e incident to x. Adding e to T obviously cannot create any circuits. This contradicts the maximality of T, and proves that it is spanning. Now suppose that T is not connected. Let T' be a component of T,

let $x \in V(T')$ and let $y \notin V(T')$. Now since G is connected, there is a path from x to y in G. Let e be the first edge of this path with exactly one end in T'. Clearly adding e to T creates no circuit, again contradicting maximality. This completes the proof of (a).

Now consider (b). If every vertex of G is incident to at least two edges in T, that is, has *degree* at least 2 in T, then it is easy to see that T contains a circuit (see Exercise 1). Hence, there is some vertex incident to exactly one edge of T. But deleting this vertex and the incident edge, we obtain a spanning tree of a graph with one less vertex. The result now follows by induction. □

For a subset of edges F of a graph G, let $G(F)$ denote the subgraph of G *induced* by F, that is, the subgraph with edges F and vertices exactly those vertices incident to some edge in F.

Corollary 2.1.2 *Let F be a subset of edges of a graph G. Then the number of edges in any maximal forest of $G(F)$ is exactly $|V(G(F))|$ minus the number of components of $G(F)$. Hence, for the independence system given by the edge-sets of the forests of a graph, r_I and r are identical. It follows that Algorithm 1.2.3 (the greedy algorithm) solves the problem of finding a maximum-weight forest.* □

This greedy algorithm for the MST problem is now generally attributed to O. Borůvka (1926) ["On a minimal problem," (in Czech) *Práce Moravske Přirodověcke Společnosti Brně* **3**], although for years the earliest reference was thought to be J. B. Kruskal (1956) ["On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society* **7** 48–50].

We postpone a discussion of the theoretical efficiency of MST algorithms until the end of the next section.

2.2 Prim's Algorithm

The main algorithm of this section, Prim's algorithm, is more efficient than the greedy algorithm on "dense graphs," simple graphs with $O(|V|^2)$ edges. Our derivation of this algorithm is based on three simple graph-theoretic lemmas. The proofs are typical of this subject: They are much easier to picture than to write down.

Lemma 2.2.1 *Let C_1, C_2 be distinct circuits of a graph G, and let $e \in C_1 \cap C_2$. Then $(C_1 \cup C_2) \backslash \{e\}$ [1] contains a circuit.*

[1] Even though 'path', 'circuit' and 'tree' have been defined in §1.1 as made up of vertices *and* edges, it is frequently convenient, as it is here, to consider them to be simply subsets of edges. We do this whenever convenient, without further comment.

Proof. C_1 and C_2 distinct implies there is an edge $f \in C_1 \backslash C_2$. Let $f = xy$. Traversing C_1 starting at y and moving away from x, we must find some first vertex y' in common with C_2 (since C_1 and C_2 have a common edge). Let P_y be the path so constructed. Similarly, construct x' and P_x. Now $x' \neq y'$, for otherwise C_1 and C_2 have at most one vertex in common. Let Q be the path in C_2 between x' and y' not including e. Then $\{f\} \cup P_x \cup P_y \cup Q$ is the desired circuit. □

Lemma 2.2.2 *Let G be a connected graph, and let T be a spanning tree of G. Then for any edge $e \in E(G) \backslash T$, $T \cup \{e\}$ contains a unique circuit, denoted $C(T, e)$, and called the* fundamental circuit *of T at e. For any edge $f \in C(T, e)$, $(T \cup \{e\}) \backslash \{f\}$ is a spanning tree.*

Proof. Let $e = xy$. Since T is spanning, it includes x and y, and since T is connected, it includes a path P joining x and y. But then $P \cup \{e\}$ is clearly a circuit, proving that $T \cup \{e\}$ contains at least one circuit. Suppose that this circuit is not unique. Then there are two distinct circuits $C_1, C_2 \subseteq T \cup \{e\}$. But then by Lemma 2.2.1, $(C_1 \cup C_2) \backslash \{e\}$ contains a circuit contained in T, a contradiction. Hence, the circuit $C(T, e) = P \cup \{e\}$ is unique.

Now consider $f = uv \in C(T, e)$, and let $T' = (T \cup \{e\}) \backslash \{f\}$. If $e = f$, then evidently $T' = T$ is a spanning tree. Assume $e \neq f$. Clearly T' contains no circuit, by the uniqueness of $C(T, e)$. Every vertex other than u and v is certainly incident to some edge of T', since it is incident to some edge of T, and u and v are each incident to some edge in $C(T, e) \backslash \{f\}$. Finally, note that T' is connected since every path in T between two vertices either does not include f, in which case it is a path in T', or it does include f, in which case replacing f by the path $C(T, e) \backslash \{f\}$ yields a (not necessarily simple) path in T'. □

Definition 2.2.3 Let $G = (V, E)$ be a graph, and let $X \subseteq V$. Define $\delta(X)$ to be the set of all edges in G with one end in X and the other in $V \backslash X$. Subsets of edges of the form $\delta(X)$ are called *cuts*. □

Lemma 2.2.4 *Let C^* be a cut and C a circuit of some graph $G = (V, E)$. Then $|C^* \cap C| \neq 1$.*

Proof. Suppose $e \in C^* \cap C$ and $e = xy$. Now $C^* = \delta(X)$ for some $X \subseteq V$, and we may assume $x \in X$. But then by definition $y \in V \backslash X$. Consider the path $C \backslash \{e\}$. Traversing it starting at x, there must be some last vertex $x' \in X$. The next edge in this path then has one end in X and one end in $V \backslash X$, and is the required second edge in $C^* \cap C$. □

The next result is the key to the spanning tree problem.

Theorem 2.2.5 *Let $G = (V, E)$ be a connected graph with edge weighting w, and let F be the edge-set of a forest in G. Let $V_1, \ldots, V_k$ be a list of the vertex-sets of the components of the edge-induced subgraph $G(F)$. Suppose for some j $(1 \leq j \leq k)$, that e is a minimum-weight edge in $\delta(V_j)$. Then among all spanning trees that are minimum-weight extensions of F, there is one containing e.*

Remark: (a) We have previously proved that all maximal forests of a connected graph are spanning trees. Thus, F is contained in some spanning tree.

(b) The above result applies to *any* forest F. F need not be contained in any minimum spanning tree of G. One can imagine applications in which, because of certain "external constraints," some edges are required to be in the solution, even though they are in no globally-minimum tree. □

Proof of Theorem 2.2.5. Given Lemmas 2.2.2 and 2.2.4, the proof is straightforward. Let T be a minimum-weight extension of F. If $e \in T$, we are done. Suppose not. Let $C^* = \delta(V_j)$, and let $C = C(T, e)$ be the fundamental circuit of T at e. Then $e \in C^* \cap C$, and so by Lemma 2.2.4, there is a second edge $f \in C^* \cap C$. Let $T' = (T \cup \{e\}) \backslash \{f\}$. Then T' is a spanning tree by Lemma 2.2.2, and

$$w(T') = w(T) + w(e) - w(f) \leq w(T),$$

by the choice of e. This completes the proof. □

The validity of Prim's algorithm, given below, is immediate from Theorem 2.2.5. This motivates, in part, the statement of the theorem. However, this result can also be used to prove the validity of the greedy algorithm, Algorithm 1.2.3. In this context it is most natural to consider an alternate, minimization version of the greedy algorithm in which one does not stop with a forest, but continues through all edges until a minimum spanning tree is constructed. The proof that the resulting tree is indeed minimum proceeds by induction. Clearly the empty forest is contained in a minimum tree. Assume, as the inductive step, that the forest constructed up to some point in the algorithm is contained in some minimum tree. We need only prove that this assumption is preserved when an edge is added to the forest. But when an edge is added it is, by the ordering (now from minimum- to maximum-weight edge), the minimum-weight edge then available that does not create a circuit together with previously chosen edges, that is, it is a minimum-weight edge with ends in different components of the current forest. Hence, by Theorem 2.2.5, the inductive assumption is preserved.

Algorithm 2.2.6 Prim's Algorithm

Input: A connected graph $G = (V, E)$ with edge weighting w.

Output: A minimum spanning tree T of G.

```
begin
    X := {v} for some v ∈ V;
    T := ∅;
    while X ≠ V do begin
        find e ∈ δ(X) such that w(e) = min_{f∈δ(X)} w(f);
        T := T ∪ {e};
        X := X ∪ {y} where e = xy and x ∈ X;
    end
end
```

Theorem 2.2.7 *Algorithm 2.2.6 is correct.* □

We conclude this section with a discussion of the computational complexity of Prim's algorithm. We do not include full details. These can for the most part be found among the references given in the bibliography. For this discussion we assume that $G = (V, E)$ is a connected graph, $n = |V|$ and $m = |E|$. We also assume that E is stored so that the ends of any edge can be found in time $O(1)$.

$O(mn)$: It is trivial to give an $O(mn)$ implementation. To store X, simply keep a bit representation, that is, keep an array **COMP**, say, of length n in which the entry for a vertex is 1 if that vertex is in X, and 0 otherwise. On each execution of the **while** loop, we find e by scanning all edges, and checking the location of its ends using **COMP**. The scan takes time $O(m)$. Updating **COMP** and T is $O(1)$. Since the **while** loop is executed $n - 1$ times, the overall bound is clearly $O(mn)$, or $O(n^3)$ for dense graphs.

$O(n^2)$: A shortcoming of the above implementation is that too much time is spent finding the edge e in the **while** loop. This situation may be improved by keeping appropriate information for the vertices in $V \backslash X$. Create arrays **SMALL** and **SMALL_EDGE**, each of length n, and initialize them as follows. For each $u \in V \backslash \{v\}$, **SMALL[u]** $= w(vu)$ if vu exists, and $= +\infty$ otherwise; **SMALL_EDGE[u]** is a pointer to v in the first case, and null otherwise. This initialization takes time $O(n)$. Now to find e we simply scan **SMALL**, and take the minimum value that arises. If **SMALL[u]** is this minimum, then **SMALL_EDGE[u]** is used to update T. This all takes time $O(n)$, as does updating these two structures.

$O(m \log n)$ [2]: This bound is not as good as the above one for dense graph. However, in some practical problems m is roughly linear in n, and then the bound is better. We give two approaches that achieve this bound. The first is more direct.

(a) We use basically the same approach used to obtain the $O(n^2)$ bound, but now we keep a (simple) heap (see [1]) in order to quickly find the minimum element in **SMALL**. The computation of the minimum is then $O(1)$. Extra work is however required to update the heap. The initial heap is constructed in time $O(n)$—that this is possible is a standard result for heaps. The updates proceed as follows. When a new vertex y is added to X, as many as $d(y)$ (the

[2] $\log n$ stands for $\log_2 n$.

degree of y) values in **SMALL** could change. Updating the heap for each of these requires time $O(\log n)$, and so the total work to add y to X is $O(d(v)\log n)$. Since $\sum_{v\in V} d(v) = 2m$, the $O(m\log n)$ bound follows.

(b) This bound is derived in [9]. It involves a substantial modification of the algorithm as stated, one that more fully makes use of the generality of Theorem 2.2.5. We keep, instead of a single vertex set X, a collection of disjoint vertex sets $X_1, \ldots, X_k$, for some k. Initially $k = n$ so that each X_j is a single vertex. At a general iteration we find, for each X_j, a minimum-weight edge in $\delta(X_j)$. This can be done in a relatively obvious way in one pass through E. Let A be the list of edges so generated. Clearly $|A| \geq k/2$, since every X_j is incident to at least one member of A, and each edge of A is incident to at most two X_j. Now by repeated application of Theorem 2.2.5, we may add all the edges of A to T. The X_j must then be updated. This takes time $O(n)$, and thus the whole iteration takes time $O(m)$. Now, since $|A| \geq k/2$, k is reduced by at least half at the next iteration. Thus, the number of iterations is bounded by $\log n$, and the overall bound follows.

$O(m + n\log n)$: This bound is due to Fredman and Tarjan. It is achieved using of Fibonacci heaps, a discussion of which is beyond the scope of these notes [M. L. Fredman and R. E. Tarjan (1984), "Fibonacci heaps and their uses in improved network optimization algorithms," *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science* 338–346]. They also obtain a bound of the form $O(m\beta(m,n))$, where

$$\beta(m,n) = \min\{j : \log^{(j)} n \leq m/n\},$$

and $\log^{(j)} n$ denotes the j^{th} iterated logarithm of n.

A difficulty with these bounds is that Fibonacci heaps are difficult to implement, and in practice do not seem to be as efficient as other theoretically less efficient kinds of heaps. Successive attempts to remedy this situation are made in [D.D. Sleator and R.E. Tarjan (1983), "Self-adjusting binary trees," *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 235–245; D. D. Sleator and R. E. Tarjan, "Self-adjusting heaps," *SIAM Journal on Computing*, to appear; R. E. Tarjan, D. D. Sleator, M. L. Fredman and R. Sedgewick, "The pairing heap: A new form of self-adjusting heap," AT&T Bell Laboratories Technical Report, September 18, 1985]. However, the corresponding theoretical bounds have not yet been obtained for these modifications.

Exercises

2.1 For sets X and Y define

$$X\Delta Y = (X\backslash Y) \cup (Y\backslash X).$$

$X\Delta Y$ is called the *symmetric difference* of X and Y. Let C' and C'' be two circuits of a graph G.

(a) Let H be a subgraph of G in which every vertex has even degree. Show that if H has at least one edge, then it contains a circuit.

(b) Show that there are edge-disjoint circuits $C_1, \ldots, C_m$ ($m \geq 1$), such that $C'\Delta C'' = C_1 \cup \ldots \cup C_m$. (This result strengthens Lemma 2.2.1.)

2.2 Prove the validity of the following algorithm:

Algorithm. Dual Greedy Algorithm.

Input: A connected graph $G = (V, E)$ with edge weighting w.

Output: A minimum spanning tree T of G.

begin
  Sort E so that $w(e_1) \geq \ldots \geq w(e_{|E|})$;
  $T := E$;
  for $j := 1$ **until** $|E|$ **do**
    if $T \backslash \{e_j\}$ is connected **then** $T := T \backslash \{e_j\}$;
end

2.3 Let G be a simple graph with $n = |V| \geq 3$. A *Hamiltonian circuit* or *tour* in G is a circuit with n vertices. G is *Hamiltonian* if is has a tour. It is easy to give examples of graphs that are not Hamiltonian, but one would expect that, if a graph has enough edges, then it is very likely Hamiltonian. Use the following steps to prove that, if every vertex of G has degree at least $n/2$, then G is indeed Hamiltonian.

(a) Let $P = (v_0, \ldots, v_k)$ be a simple path in G, and assume that neither v_0 nor v_k is joined by an edge to a vertex not in P (thus, P cannot be "extended" to a longer path). Prove, using a counting argument and the fact that $d(v_0) + d(v_k) \geq n$, that for some j, $v_0 v_{j+1}$ and $v_j v_k$ are edges of G (draw a picture!). Note that deleting $v_j v_{j+1}$ and adding these two edges to P yields a circuit.

(b) Let C be a circuit with fewer than n vertices. Use a counting argument, using the fact that either C or its complement has at most $n/2$ vertices, to show that some vertex in G is joined to a vertex not in C. Thus, by deleting an appropriate edge of C we may "extend" C to a longer path.

Can this proof be made into an algorithm? If so, can you estimate its complexity?

2.4 Show that the symmetric difference of two cuts is a disjoint union of cuts.

Chapter 3

Shortest Paths

3.1 Introduction

The shortest path problem is one of the most-studied problems in combinatorial optimization, and could easily be the subject of several chapters. Among its variations are the k-shortest path problem, the longest path, most-dependable path and maximum capacity path problems, and the problem of finding shortest paths with an even or odd number of edges. Our treatment will, however, be rather brief. We consider only the problem of finding shortest paths from a single given vertex to all other vertices. For this version we examine a linear-programming (LP) [1] approach (the Ford algorithm), the Moore-Bellman algorithm, and Dijkstra's algorithm.

Those wishing a more detailed treatment are referred to [6]. For additional material see also:

- Domschke, K. (1972), "Kürzeste Wege in Graphen," *Mathematical Systems in Economics* **2**, Verlag A. Hain, Meisenheim am Glan
- Dreyfus, S. E. (1969), "An appraisal of some shortest-path algorithms," *Operations Research* **17** 395–412 (a classic article)
- Glover, F., D. Klingman and Phillips (1985), "A new polynomially bounded shortest path algorithm," *Operations Research* **33** 65–73 (this paper discusses the more recent literature)
- Syslo, M. M., N. Deo and J. S. Kowalik (1983), the bibliography of *Discrete Optimization Algorithms, with Pascal Programs*, Prentice-Hall, Englewood Cliffs, New Jersey

[1] A very brief introduction to linear programming is given in §4.1

3.2 Definitions

It is customary to study shortest-paths in the context of directed graphs. We begin with a short introduction to these.

A *directed graph*, or *digraph*, $D = (V, A)$ is an ordered pair made up of a finite set V of *vertices* and a finite set A of *arcs*, such that each arc has associated with it an ordered pair of (not necessarily distinct) vertices called its *tail* and its *head*, respectively. For an arc e, we denote the tail by $t(e)$ and the head by $h(e)$. $t(e)$ is then a *predecessor* of $h(e)$ and $h(e)$ a *successor* of $t(e)$. When no confusion can arise (for example, when there are no similarly directed parallel arcs) we write $e = (t(e), h(e))$.

All the notions previously introduced for (undirected) graphs, such as tree, path and circuit, can be applied to digraphs by considering the underlying graph, simply ignoring the directions on the arcs. For some of these notions we also define corresponding directed versions. Thus, a $v_0 \rightarrow v_k$ *dipath* $P = (v_0, \ldots, v_k)$ is a path such that (v_{i-1}, v_i) is and arc $(i = 0, \ldots, k)$. *Dicircuit* is defined similarly. For $X \subseteq V$, $\delta^-(X) = \{e : t(e) \in X, h(e) \in V \backslash X\}$ and $\delta^+(X) = \delta^-(V \backslash X)$ (see Definition 2.2.3).

Definition 3.2.1 Let $D = (V, A)$ be a digraph with a distinguished vertex r, called the *root*, and a weight or *length* [2] function w defined on A. For a dipath $P = (v_0, \ldots, v_k)$, the *length of* P, $w(P)$, is defined by $w(P) = \sum_{i=0}^{k} w(v_{i-1}, v_i)$. The *shortest path problem*, SPP, for (D, w, r) is to find, for each vertex v of D, a minimum-length, or *shortest*, dipath from r to v.

It may seem that a more natural version of the SPP would be one in which a single dipath between a pair of specified vertices is the goal. Indeed, this form of the problem is discussed in some of what follows. However, most of the practical *algorithms* that solve this "more natural" problem, also solve the one in the definition, and so this version will be the focus of our treatment.

Finally, one can also define the shortest path problem in an obvious way for undirected graphs. Instances of this problem can be converted to a directed problem by replacing each edge by two oppositely directed arcs, both having the same length as the original edge. If the length on the edge is nonnegative, then the techniques given in this chapter are applicable to the resulting directed problem in a straightforward way. However, if there are negative-length edges, then a more sophisticated approach using "nonbipartite matching" is called for. This approach is discussed in [6].

[2] The reader is cautioned that these "lengths" need not be actual physical lengths, and so may be negative. For this reason the use of the word 'weight' here might have some advantages, but we prefer the more natural 'length'.

3.3 Minty's Analog Algorithm and an LP Approach

George Minty suggested the following analog approach. Let $G = (V, E)$ be a graph with two distinguished vertices r and s and with a nonnegative length function w defined on the edges. Construct a "string model" of G in which the edges have actual lengths proportional to their length as defined by w. Now, grasp this model by its two "ends," that is, by the two vertices r and s, and pull. One would expect that the length of a shortest path between r and s will then be the distance by which they are separated after this pulling. In fact, some additional reflection reveals that this expectation is not quite justified, since some of the strings may be "caught" in loops with others, but that some retying of the strings will remove these loops and give the desired result.

The relevance of Minty's method here is: It suggests that we can solve the SPP, which is a minimization problem, in a natural way as a maximization problem. The formulation below uses this *dual* approach.

Let $D = (V, A)$ be a digraph with length function w. Let r and s be two distinguished vertices of D. Consider the following linear program:

$$\begin{array}{ll} \max & u(s) - u(r) \\ \text{s.t.} & u(h(e)) - u(t(e)) \leq w(e) \quad (e \in A) \end{array} \tag{3.1}$$

To help understand (3.1), suppose that there exists an $r \rightarrow v$ dipath for each $v \in V$, and assume that D contains no negative (-length) dicircuit. Let $u(v)$ be the length of a shortest $r \rightarrow s$ dipath for each $v \in V$. Note that we may assume any such dipath is simple, since otherwise it contains a dicircuit, and any dicircuit can be deleted because, by assumption, it has nonnegative length. Now clearly $u(r) = 0$, and so $u(s) - u(r)$ is evidently the length of a shortest $r \rightarrow s$ dipath. Let $e \in A$, and let P be a shortest $r \rightarrow t(e)$ dipath. Appending e to P we obtain an $r \rightarrow h(e)$ dipath P' (not necessarily simple); moreover,

$$u(h(e)) \leq w(P') = w(P) + w(e) = u(t(e)) + w(e).$$

That is, $u(h(e)) - u(t(e)) \leq w(e)$. Hence, if D contains no negative dicircuit, then (3.1) is feasible.

Theorem 3.3.1 *If (3.1) has an optimal solution u^*, then $u^*(s) - u^*(r)$ is the length of a shortest $r \rightarrow s$ dipath.*

Proof. Let $P = (v_0, \ldots, v_k)$ be a $v_0 \rightarrow v_k$ dipath. Then we have

$$\begin{aligned} u^*(v_k) - u^*(v_0) &= \sum_{j=1}^{k} (u^*(v_j) - u^*(v_{j-1})) \\ &\leq \sum_{j=1}^{k} w(v_{j-1}, v_j) \\ &= w(P). \end{aligned}$$

It follows that for any vertices x and y, and any $x \rightarrow y$ dipath P, $u^*(y) - u^*(x)$ is a lower bound on $w(P)$. Note also that if u^* is *tight* for each arc in P, that is, if $u^*(h(e)) - u^*(t(e)) = w(e)$ for each arc e of P, then $u^*(y) - u^*(x) = w(P)$.

Now let X be the set of all vertices reachable from r by u^*-tight dipaths. Clearly $r \in X$, and if $s \in X$, then it follows by the computations of the previous paragraph that $u^*(s) - u^*(r)$ is the length of a shortest $r \rightarrow s$ dipath, as required. Assume $s \notin X$.

Let $\alpha = \min\{w(e) - [u^*(h(e)) - u^*(t(e))] : e \in \delta^-(X)\}$. Then $\alpha > 0$, by the choice of X. For each $v \in V \backslash X$, define $u'(x) := u^*(x) + \alpha$, and for $x \in X$ define $u'(x) := u^*(x)$. The constraints of (3.1) are then affected only for arcs e with at least one end in $V \backslash X$. By the choice of α, they are still valid on $\delta^-(X)$, and they are obviously unaffected for any arcs with both ends in $V \backslash X$. It remains only to consider arcs $e \in \delta^+(X)$. But for any such arc we have $u'(h(e)) - u'(t(e)) = u^*(h(e)) - u^*(t(e)) - \alpha \leq w(e)$. Since $u'(s) - u'(r) = u^*(s) - u^*(r) + \alpha > u^*(s) - u^*(r)$, this contradicts the optimality of u^*. □

The question of existence of an optimal solution for (3.1) is answered by the following theorem.

Theorem 3.3.2 *LP (3.1) is feasible iff D has no negative dicircuits. If (3.1) is feasible, it is bounded iff there exists at least one $r \rightarrow s$ dipath.*

Proof. In the proof of Theorem 3.3.1 it is shown that the length of any $r \rightarrow s$ dipath is a bound on $u(s) - u(r)$, for any feasible solution u of (3.1). Conversely, suppose that there exists no $r \rightarrow s$ dipath. Let X be the set of all vertices reachable by dipaths from r. Then we have $s \notin X$. Now given any feasible solution u of (3.1), we can clearly add an arbitrary constant to $u(x)$ for each $x \in V \backslash X$, not affecting feasibility, and increasing the value of the objective $u(s) - u(r)$ without bound. Thus, (3.1) is unbounded. This proves the boundedness criterion stated in the theorem.

Now consider the feasibility question. Suppose $C = (v_0, \ldots, v_k = v_0)$ is a negative dicircuit. Then the constraints of (3.1) imply

$$\begin{aligned} 0 &= u(v_k) - u(v_0) \\ &= \sum_{j=1}^{k} (u(v_j) - u(v_{j-1})) \\ &\leq \sum_{j=1}^{k} w(v_{j-1}, v_j) \\ &= w(C) \\ &< 0, \end{aligned}$$

a contradiction. It follows that the existence of negative dicircuit precludes the existence of a feasible solution for (3.1).

Finally, consider the converse. Suppose that D has no negative dicircuit, and let X be the set of all vertices x such that an $r \rightarrow x$ dipath exists. Let $u(x)$ be the length of a shortest $r \rightarrow x$ dipath for each $x \in X$. By the discussion following (3.1), $u(x)$ is well defined and satisfies the constraints of (3.1) on X. If $X = V$, we are done; otherwise, pick $r' \in V \backslash X$, and repeat the above construction: Let X' be the subset of vertices reachable from r', and define u' appropriately on X'. The constraints of (3.1) are then clearly satisfied on X' by u'. We would now like to extend u' to $X \cup X'$, but there is a difficulty: Even though there are, by definition, no arcs from X' to $X \backslash X'$, there may be arcs from $X \backslash X'$ to X'. This situation is remedied by choosing an appropriately large constant and adding it to u on $X \backslash X'$. Now setting $u' := u$ on $X \backslash X'$, we obtain a feasible solution on $X \cup X'$. Repeating this entire construction until a solution is found for all of V completes the proof. □

The above feasibility condition, that negative dicircuits do not exist, is fundamental. The SPP, though generally considered well solved, is in fact $\mathcal{NP}$-hard for digraphs with negative dicircuits: A solution to this problem is easily seen to imply a solution of the TSP.

3.4 Solving (3.1)

Obviously, (3.1) can be solved using the simplex method, or any other method for solving LPs. However, the problem is simpler than that. L. R. Ford, Jr., offered the following direct method.

Algorithm 3.4.1 Ford's Algorithm.

Input: An SPP (D, r, w) with no negative dicircuits.

Output: An optimal solution u of (3.1).

Comment: For any x, if no $r \rightarrow x$ dipath exists, the algorithm terminates with $u(x) = +\infty$. This is consistent with Theorem 3.3.2.

begin
  $u(r) := 0$;
  for $x \in V \backslash \{r\}$ **do** $u(x) := +\infty$;
  while $u(h(e)) - u(t(e)) > w(e)$ for some $e \in A$ **do**
    $u(h(e)) := u(t(e)) + w(e)$;
end

As is noted again below, Ford's algorithm is not polynomial (and is not even finite if we allow negative dicircuits). However, it is instructive, and at least convenient for hand computations. Note also that the algorithm does not actually output the shortest paths, when they exist, but simply their lengths. However, it is not hard to see that these can be found by appropriately recording the arcs e for which $u(h(e))$ changes in the **while**.

Theorem 3.4.2 *Ford's algorithm is finite; moreover, if for a particular $x \in X$, $u(x) < +\infty$ at termination, then $u(x)$ is the length of a shortest $r \rightarrow x$ dipath, and if $u(x) = +\infty$, then there is no $r \rightarrow x$ dipath.*

Proof. We give only an outline. The key is to prove that whenever $u(x)$ is finite for a particular vertex x, then $u(x)$ is the length of *some* simple $r \rightarrow x$ dipath. This is certainly true at the start of the algorithm. One needs only prove that the property is maintained by the assignment statement in the **while**.

Now, given the above fact about u, to see that the algorithm is finite, observe that each update strictly decreases some $u(x)$. But there are only a finite number of simple $r \rightarrow x$ dipaths, and hence only a finite number of possible values for $u(x)$.

Finally, the fact that the $u(x)$ at termination have the correct values follows because, first, when finite they do correspond to the length of some dipath, and, second, the constraints of (3.1), which are evidently satisfied at termination, imply that $u(x)$ is a lower bound for the length of any $r \rightarrow x$ dipath. □

Ford's algorithm has two major shortcomings: It can only be applied to digraphs that are known not to contain negative dicircuits, and it has exponential worst-case behavior. We have not explicitly demonstrated this second claim, but it is not hard to do so. That this is possible should, in any case, not be surprising since the order in which updates occur in the algorithm is completely arbitrary. The *bound* given in the proof is certainly bad.

Both the above difficulties are dealt with by the Moore-Bellman algorithm. This algorithm works roughly as follows. To start, an ordering of the vertices is fixed. Then repeated passes are made through the vertices, using this ordering, $|V|$ passes in total. Each time a vertex is encountered during a pass, an "update" is performed

on all arcs with tail equal to that vertex. Thus, on each pass each arc e is examined exactly once. The total work for this procedure is clearly polynomial.

Algorithm 3.4.3 Moore-Bellman Algorithm

Input: A digraph $D = (V, A)$ with arc lengths w. We assume $V = \{1, \ldots, n\}$ and $r = 1$.

Output: The conclusion that D has a negative dicircuit, or vertex numbers $u^n(1)$, $\ldots$,$u^n(n)$ with the following interpretation: If $u^n(j) < +\infty$, then $u^n(j)$ is the length of a shortest $1 \rightarrow j$ dipath; otherwise, if $u^n(j) = +\infty$, there is no $1 \rightarrow j$ dipath. In addition, if there is no negative dicircuit, then for each $u^n(j) < +\infty$, $j \neq 1$, a vertex PRED[j] is output such that (PRED[j],j) is the last arc in some shortest $1 \rightarrow j$ dipath.

```
begin
    u^0(1) := 0;
    for j := 2,...,n do u^0(j) := +∞;
    for i := 1,...,n do begin
        for j := 1,...,n do u^i(j) := u^{i-1}(j);
        for j := 1,...,n do begin
            for (j,k) ∈ A do begin
                if u^i(k) > u^{i-1}(j) + w(j,k) then begin
                    u^i(k) := u^{i-1}(j) + w(j,k);
                    PRED[k] := j;
                end
            end
        end
    end
    if u^n(j) < u^{n-1}(j) for some j then
        print "There exists a negative dicircuit";
end
```

Theorem 3.4.4 *The Moore-Bellman shortest path algorithm is correct and runs in time* $O(|V||E|)$.

Proof. Again, we give only an outline, and, again, the key to the proof is an appropriate interpretation of u. In this case one can show that when $u^i(j) < +\infty$ for some vertex j and some $i = 0, \ldots, n$, then $u^i(j)$ is the length of a shortest $1 \rightarrow j$ dipath using at most i arcs. Certainly this is true at the start of the algorithm, after u^0 is initialized, and it is straightforward to verify the claim in general, by induction on i. Now, note that no simple dipath can have more than $n - 1$ arcs, since D has only n vertices. Hence, if $u^n(j) < u^{n-1}(j)$ for some j, then by the above claimed interpretation of $u^n(j)$, $u^n(j)$ must be realized by a nonsimple dipath, and this dipath must contain a negative dicircuit. The remaining parts of the proof are routine. □

Note that Algorithm 3.4.3 is not guaranteed to find a negative dicircuit whenever one exits, only to find one if it stands in the way of finding all shortest dipaths starting at the root. In particular, if there is a negative dicircuit, but no vertex on it is reachable from the root, then the algorithm will *not* find this dicircuit. Note also that the implementation suggested above admits some obvious simplifications. For notational convenience we have written the u values in such a way that a total storage requirement of $O(n^2)$ is suggested: $u^0(1), \ldots, u^n(n)$. However, the algorithm never requires knowledge of more the $O(n)$ of these terms: $u^{i-1}(j)$ and $u^i(j)$ for $j = 1, \ldots, n$ and the current i. A second simplification, or at least speedup, can be obtained by better exploiting the information in the algorithm. In particular, while the algorithm uses u^{i-1} in the updates for u^i, for a given pass i, it could clearly only help to use the current values of u^i where these are smaller than u^{i-1}. This idea leads to a theoretically faster algorithm [Yen (1970), "An algorithm for finding shortest routes from all source nodes to a given destination in general network," *Quarterly Journal of Applied Mathematics* **27** 526–530; see also [6], pages 76 and 77], but does have the disadvantage that the proof is somewhat more involved: The interpretation of $u^i(j)$ given in the proof is no longer quite correct.

3.5 Some Miscellaneous Results

We describe here one result in some detail, Dijkstra's algorithm, and mention two others: the SPP for acyclic digraphs, and the all-pairs problem.

Dijkstra's algorithm [Dijkstra (1959), "A note on two problems in connexion with graphs," *Numerische Mathematik* **1** 269–271] seems to have been frequently discovered, and is certainly frequently used, especially in theoretical applications. The reader will almost certainly encounter it, and so it seems wise to present it here. The situation treated is that in which all arc lengths are nonnegative. Thus, the problem of negative dicircuits is automatically settled. The procedure itself can be viewed as a "label fixing" procedure, rather than a "label adjusting" procedure as in the case of the Ford and Moore-Bellman algorithms. The reason for this designation can be seen in the following description of the algorithm.

The algorithm begins by assigning the root the label 0 and designating this label as "fixed;" 0 *is* clearly the length of the shortest path from the root to itself. All other vertices receive as "temporary" labels the length of the arc from the root to that vertex, if there is one, and otherwise the temporary label $+$. Among the vertices with temporary labels the vertex x with the smallest label is then selected, and is designated as fixed. This label must be the length of a shortest path by virtue of the nonnegativity of the arc lengths. Now, for each vertex y with a temporary label that is also a successor of x, we compare the temporary label with the label of x plus $w(x, y)$, and update if necessary. Then we fix the label of the vertex with

the smallest temporary label, and so on.

Algorithm 3.5.1 Dijkstra's Algorithm.

Input: A SPP (D, r, w). Assume $w \geq 0$.

Output: Vertex numbers $u(x)$ $(x \in V)$ with the following interpretation: If $u(x) < +\infty$, then $u(x)$ is the length of a shortest $r \rightarrow x$ dipath; otherwise, if $u(x) = +\infty$, there is no $r \rightarrow x$ dipath. In addition, for each $u(x) < +\infty$, $x \neq r$, a vertex PRED[x] is output such that (PRED[x],x) is the last arc in some shortest $r \rightarrow x$ dipath.

```
begin
   TEMP := V;
   u(r) := 0;
   for x ∈ V\{r} do u(x) := +∞;
   while there exists x ∈ TEMP with u(x) < +∞ do begin
      select x ∈ TEMP such that u(x) := min_{y∈TEMP} u(y);
      TEMP := TEMP\{x};
      for (x,y) ∈ A and y ∈ TEMP do
         if u(y) > u(x) + w(x,y) then begin
            u(y) := u(x) + w(x,y);
            PRED[y] := x;
         end
   end
end
```

Theorem 3.5.2 *Dijkstra's algorithm is correct and runs in time* $O(|V|^2)$.

Remark: A $O(|E| + |V| \log_2 |V|)$ implementation of Dijkstra's algorithm is given in [M. L. Fredman and R. E. Tarjan (1984), "Fibonacci heaps and their uses in improved network optimization algorithms," *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, 338–346]. See the discussion of an $O(m + n \log_2 n)$ algorithm for minimum spanning trees at the end of Chapter 2.

Proof. Denote $F = V \backslash$TEMP. Note that at any stage in the algorithm and for $y \neq r$: $u(y) = \min\{u(x) + w(x,y) : x \in F, (x,y) \in A\}$, and PRED[$y$] $\in F$ if $u(y) < +\infty$.

The main step of the proof is to show that at any stage of the algorithm and for any $y \in F$, $u(y)$ is the length of a shortest $r \rightarrow y$ dipath. This is trivially true initially since $F = \emptyset$. Assume inductively that it is true at some stage, and suppose x is the vertex selected for inclusion in F in the next application of the **while**. If $x = r$, $u(r) = 0$ is clearly the length of some $r \rightarrow r$ dipath; otherwise, let $y =$ PRED[x] $\in F$ and let P be a shortest $r \rightarrow y$ dipath. Write $P(y,x)$ for the dipath P with arc (y,x) appended. Then we have

$$u(x) = u(y) + w(y,x) = w(P) + w(y,x) = w(P(y,x))$$

where the first equality follows by the definition of PRED[x] and the second by the inductive hypothesis on F. Thus, $u(x)$ is the length of some $r \rightarrow x$ dipath.

Now suppose Q is another $r \rightarrow x$ dipath, let z be the last vertex of Q in F, let Q_1 be the $r \rightarrow z$ portion of Q, let e be the next arc, and let Q_2 be the remaining portion of Q. Then

$$w(Q) = w(Q_1) + w(e) + w(Q_2) \geq u(z) + w(e) \geq u(x),$$

where the first inequality follows by the inductive hypothesis and the assumption that $w(Q_2) \geq 0$, and the second inequality by the choice of x. Combining this result and the result of the previous paragraph we conclude that $u(x)$ is the length of a shortest $r \rightarrow x$ dipath.

To complete the proof is now straightforward. The asserted property of PRED in output is implicit in the calculation of the second paragraph of the proof. To deduce the required properties of u note first that $u(x) = +\infty$ at termination iff $x \in$ TEMP. Hence, the above proved property of F yields the desired properties of $u(x)$ when $u(x) < +\infty$. In the remaining case, if $u(x) = +\infty$ for some x, then necessarily $\delta^-(F) = \emptyset$ at termination, for otherwise $u(y) = \min\{u(x) + w(x,y) : x \in F, (x,y) \in A\} < +\infty$ for some $y \in$ TEMP. This completes the proof. □

We close, as promised, with remarks on two further special versions of the SPP. Both are discussed in [6]; the all pairs problem, the harder of the two, is also discussed in [9] (pages 129–133).

Shortest Paths in Acyclic Digraphs: Let (D, r, w) be a SPP problem and assume that D has no dicircuit. Such digraphs are called *acyclic*. Since for acyclic digraphs there are *a fortiori* no negative dicircuits (since there are no dicircuits whatsoever!), it clear that the SPP on such digraphs is tractable. Indeed, it turns out to be *very easy*. In the standard algorithms the first step is to "topologically sort" the vertices of D, that is, to find an ordering $v_1, \ldots, v_n$ ($n = |V|$) such that (v_j, v_k) is an arc only if $j < k$. It is then not hard to see that $u(v_1) = 0$ and $u(v_k) = \min\{u(v_j) + w(v_j, v_k) : j < k, (v_j, v_k) \in A\}$ ($k = 2, \ldots, n$). Solving these "equations" in a straightforward way gives the desired solution in time $O(|E|)$. This algorithm is at the heart of the subject called critical path scheduling (the probabilistic version of which is PERT). The idea of topological sort is also useful in its own right. For example, it is used in the UNIX utility 'make'.

All Pairs SPP: Given a digraph D with edge-lengths w, the problem is to find shortest $x \rightarrow y$ dipaths for all pairs of vertices x, y. Clearly, this problem can be solved directly by $n = |V|$ applications of the Moore-Bellman procedure, once for each of n different choices of root. This gives a worst-case bound of $O(n^4)$. There is, however, a better method. Using an idea of Floyd, S. Warshall showed how to solve this problem in time $O(n^3)$, the same as for the usual SPP! The algorithm is not difficult. See [6] for details.

Exercises

3.1 Let $D = (V, A)$ be a digraph with a root vertex r. A rooted arboresence of D is a spanning tree T such that for every vertex x, the unique r–x path in T is an $r \rightarrow x$ dipath.

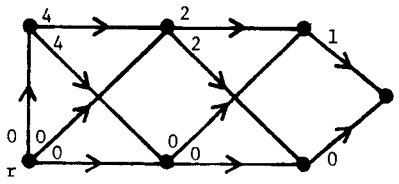
Algorithm. Simplex Algorithm for Shortest Paths.

Input: An SPP (D, r, w) and an arboresence T rooted at r. Assume D has no negative dicircuits.

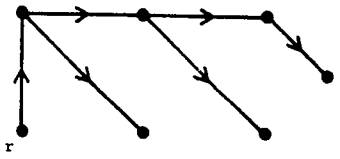
Output: An optimal solution of (3.1) (same as Ford's algorithm).

begin
  for $x \in V$ **do** $u(x) := w(P)$ where P is an $r - x$ dipath in T;
  while $u(h(e)) > u(t(e)) + w(e)$ for some $e \in A \backslash T$ **do begin**
    $f :=$ arc of T with $h(f) = h(e)$;
    $T' :=$ component of $T \backslash \{f\}$ containing $h(e)$;
    $\Gamma := u(h(e)) - u(t(e)) - w(e)$;
    for $x \in V(T')$ **do** $u(x) := u(x) - \Gamma$;
    $T := (T \backslash \{f\}) \cup \{e\}$;
  end
end

(a) Consider the following digraph:

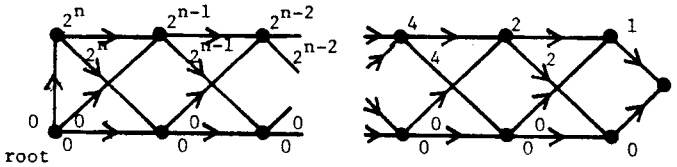


Start with the arboresence



and show that the above algorithm can take 15 iterations to solve the shortest path problem.

(b) Consider the following digraph (due to J. Edmonds):



Show that the above algorithm can take $3 \cdot 2^{n+1} - 2n - 5$ iterations to solve this problem. Deduce that it is not possible to give a polynomial time bound for Ford's Algorithm.

Chapter 4

Introduction to Polyhedral Combinatorics

4.1 Introduction

We begin with a brief introduction to linear programming and several of its associated polyhedral concepts. We then consider the traveling salesman problem, as an example of how linear programming techniques can be applied to hard combinatorial problems. This will serve to motivate some of the ideas that come in later chapters.

An excellent linear programming book is [4]. For a thorough treatment of polyhedral preliminaries see Chapters 7 and 8 of [10]; included in these chapters are proofs of (4.1.2) and (4.1.4), which are not proved in these notes.

A *linear program*, or LP, is an optimization problem of the form

$$\begin{array}{lll} & \min & c^T x \\ \text{(Primal)} & \text{s.t.} & Ax \geq b \\ & & x \geq 0 \end{array} \tag{4.1}$$

where, for some positive integers m and n, c is an $n \times 1$ column vector, b is an $m \times 1$ column vector, A is an $m \times n$ matrix and x is an $n \times 1$ column vector of variables. $c^T x$ is the *objective function* and the inequalities in $Ax \geq b$ are the *constraints*. Any $x \geq 0$ such that $Ax \geq b$ is called *feasible*. $\{x \geq 0 : Ax \geq b\}$ is the *feasible region*.

The form of (4.1) is, of course, not the most general form an LP can take. We could as well consider maximum problems and mixed constraints, including equality as well as inequality constraints. One can also consider variables with non-trivial upper and lower bounds, or variables without any bounds whatsoever, so-called *free* variables.

The *dual* of (4.1) is the LP

$$\begin{array}{lll} & \max & b^T y \\ \text{(Dual)} & \text{s.t.} & A^T y \leq c \\ & & y \geq 0 \end{array} \tag{4.2}$$

When discussing the dual, (4.1) is called the *primal* (as indicated in (4.1)). A simple but important result for dual pairs of LPs is the following.

Theorem 4.1.1 (Weak Duality Theorem) *If x and y are feasible solutions of (4.1) and (4.2), respectively, then $c^T x \geq b^T y$. In particular, if $c^T x = b^T y$, then x and y are optimal solutions of (4.1) and (4.2), respectively.*

Proof. A straightforward application of the conditions of the theorem yields

$$c^T x \geq (y^T A)x = y^T(Ax) \geq y^T b = b^T y. \quad \square$$

There are several alternative versions of this theorem based on alternate forms for the primal LP (and, hence, the dual LP). For example, if the primal has the form $\min\{c^T x : Ax = b, x \geq 0\}$, involving only equality constraints, then the dual has the form $\max\{b^T y : A^T y \leq c\}$ where the y variables are free. Examining the proof of weak duality suggests why: The second inequality in the proof is an equality, and so the nonnegativity of the y variables is not needed.

Much deeper than (4.1.1) is the following result, widely attributed to John von Neumann.

Theorem 4.1.2 (Strong Duality Theorem) *If either (4.1) or (4.2) has an optimal solution, then both have optimal solutions and the optimal values are equal.* $\square$

The strong duality theorem can be proved constructively using G. B. Dantzig's simplex algorithm. It can also be proved using a separating hyperplane theorem, such as the Farkas Lemma (see [10]).

We now introduce some of the polyhedral theory associated with LPs. As above, A is an $m \times n$ matrix, and x and b are (column) vectors of appropriate dimension. A set of the form $P = \{x : Ax \leq b\}$ is called a *polyhedron*. A set $C \subseteq \mathbf{R}^n$ is called a (convex) *cone* if for any $x, y \in C$ and any scalars $\alpha, \beta \geq 0$, $\alpha x + \beta y \in C$. Given vectors $y^1, \ldots, y^k \in \mathbf{R}^n$, and scalars $\alpha_1, \ldots, \alpha_k \geq 0$, the linear combination $\alpha_1 y^1 + \ldots + \alpha_k y^k$ of $y^1, \ldots, y^k$ is called a *positive combination*. If, in addition, $\alpha_1 + \ldots + \alpha_k = 1$, it is called a *convex combination*. For $X \subseteq \mathbf{R}^n$, $\operatorname{pos} X$ denotes the *positive hull* of X, the set of all finite positive combinations of vectors in X, and $\operatorname{conv} X$ denotes the *convex hull*, the set of all finite convex combinations of vectors in X. Clearly $\operatorname{pos} X$ is a cone. If $\operatorname{conv} X = X$, then X is *convex*. Cones and polyhedra

are examples of convex sets. If X is finite, we say that $C = \operatorname{pos} X$ is *finitely generated.* Sets of the form $\operatorname{conv} X$ where X is finite are called *polytopes.* Finally, define a point x in a convex set X to be an *extreme point* of X if $x = \alpha y + (1-\alpha)z$ for $0 < \alpha < 1$ and $y, z \in X$ implies $x = y = z$. Thus, x is extreme if it is not the convex combination of distinct other points in X.

Proposition 4.1.3 *A vector $y \in \mathbf{R}^n$ is an extreme point of $P = \{x : Ax \leq b\}$ iff $y \in P$ and y is the unique solution of some subsystem $A'x = b'$ of $Ax \leq b$.*

Proof. First we show the necessity of the condition. Let y be an extreme point of P, and let $A'x = b'$ be the maximal subsystem such that $A'y = b'$. Let $A''x \leq b''$ be the subsystem of remaining inequalities in $Ax \leq b$. If y is not the unique solution of $A'x = b'$, then there is a nonzero vector z such that $A'z = 0$ (take z to be the difference of two distinct solutions of $A'x = b'$). Since $A''y < b''$, and this system is finite, there is some $\beta > 0$ such that $Ax^j \leq b$ $(j = 1, 2)$ where $x^1 = y - \beta z$ and $x^2 = y + \beta z$. But then $y = \frac{1}{2}x^1 + \frac{1}{2}x^2$, a contradiction. Hence y is the unique solution of $A'x = b'$, as required.

Now to prove sufficiency, suppose $y \in P$ and y is the unique solution of the subsystem $A'y = b'$. Suppose $y = \alpha x + (1-\alpha)z$ where $x, z \in P$ and $0 < \alpha < 1$. Now we have

$$b' = A'y = \alpha A'x + (1-\alpha)A'z \leq \alpha b' + (1-\alpha)b' = b'.$$

Hence, $0 < \alpha < 1$ implies $A'x = A'z = b'$, and so $y = x = z$, as required. □

For sets $A, B \subseteq \mathbf{R}^n$, define $A + B = \{a + b : a \in A, b \in B\}$, the algebraic sum of A and B.

Theorem 4.1.4 (Farkas-Minkowski-Weyl) *$P \subseteq \mathbf{R}^n$ is a polyhedron iff*

$$P = \operatorname{conv} X + \operatorname{pos} Y,$$

where X and Y are finite subsets of $\mathbf{R}^n$. □

It is natural to ask when we can put special conditions on the vectors in X and Y. Define the *lineality space* of a convex set P, $\operatorname{lin} P$, by $\operatorname{lin} P = \emptyset$ if $P = \emptyset$, and otherwise $\operatorname{lin} P = \{x : y + \alpha x \in P \text{ for all } y \in P \text{ and } \alpha \in \mathbf{R}\}$. If $P = \{x : Ax \leq b\}$ is a polyhedron then it is easy to see that $\operatorname{lin} P = \{x : Ax = 0\}$. We can also prove, using (4.1.4), that P has an extreme point iff $\operatorname{lin} P = \{0\}$. Clearly $\operatorname{lin} P = \{0\}$ is necessary for this. To see the converse, assume $\operatorname{lin} P = \{0\}$, and take $y \in P$ such that the maximal subsystem $A'y = b'$ of $Ay \leq b$, satisfied at equality, has as many rows as possible. If y is not the unique solution of $A'y = b'$, and hence not an extreme point, then there is a nonzero x such that $A'x = 0$. As $\{x : Ax = 0\} = \{0\}$, $Ax \neq 0$, and so for some scalar α, $y + \alpha x \in P$ satisfies more equalities than y, a contradiction.

Proposition 4.1.5 *Let P be a nonempty polyhedron. Then $\operatorname{lin} P = \{0\}$ iff there exist finite sets X and Y, $X \neq \emptyset$, such that $P = \operatorname{conv} X + \operatorname{pos} Y$ and such that X is the set of extreme points of P.*

Proof. If P has an extreme point, then $\operatorname{lin} P = \{0\}$. This proves one direction of the proposition. To prove the the other, first apply Theorem 4.1.4 to represent P as $\operatorname{conv} X + \operatorname{pos} Y$ for finite sets X and Y. Assume that X is chosen to be minimal.

Suppose $z = x + y$ is an extreme point of P, where $x \in \operatorname{conv} X$ and $y \in \operatorname{pos} Y$. If $y \neq 0$, then $z = \frac{1}{2}x + \frac{1}{2}(x + 2y)$ and $x \neq x + 2y$, a contradiction. Hence, $z \in \operatorname{conv} X$. But then obviously $z \in X$. It follows that X contains all extreme points of P.

Now suppose $z \in X$ is not extreme and consider the following calculation. Again, write $z = x + y$, where $x \in \operatorname{conv} X$ and $y \in \operatorname{pos} Y$, and assume that $y \neq 0$. If $x \in \operatorname{conv}(X \backslash \{z\})$, clearly we may delete z from X, contradicting the minimality of X. Otherwise $x = \alpha z + (1 - \alpha)x'$ where $x' \in \operatorname{conv}(X \backslash \{z\})$ and $0 < \alpha < 1$. But then $z = x' + y/(1 - \alpha)$, and again we conclude that z may be deleted from X.

Now, using the fact that z is not extreme we have

$$z = \alpha(x^1 + \beta_1 y^1) + (1 - \alpha)(x^2 + \beta_2 y^2) \tag{4.3}$$

where $x^1 + \beta_1 y^1 \neq x^2 + \beta_2 y^2$, $0 < \alpha < 1$ and $x^j \in \operatorname{conv} X$, $y^j \in \operatorname{pos} Y$, $\beta_j \geq 0$ $(j = 1, 2)$. By the calculation of the previous paragraph we may assume $\alpha\beta_1 y^1 + (1 - \alpha)\beta_2 y^2 = 0$, which implies $\{+\beta_i y^i, -\beta_i y^i\} \subseteq \operatorname{pos} Y$ $(i = 1, 2)$, and so $\operatorname{lin} P = \{0\}$ implies $\beta_1 y^1 = \beta_2 y^2 = 0$. We conclude that $z = \alpha x^1 + (1 - \alpha)x^2$ and $x^1 \neq z \neq x^2$. This will complete the proof if we can show that $x^1, x^2 \in \operatorname{conv}(X \backslash \{z\})$. If not, suppose say $x^1 = \beta z + (1 - \beta)x^3$, where $0 < \beta < 1$ and $x^3 \in \operatorname{conv}(X \backslash \{z\})$. Then

$$z = \frac{1 - \alpha}{1 - \alpha\beta} x^2 + \frac{\alpha(1 - \beta)}{1 - \alpha\beta} x^3,$$

which is a convex combination of x^2 and x^3. Repeating this argument for x^2, if necessary, completes the proof. □

Applying (4.1.5) to linear programming, we see that if the feasible region of an LP is bounded and nonempty, then it has an extreme-point optimal solution.

4.2 The Traveling Salesman Problem

We consider here a formulation of the TSP that is superficially different from that given in Chapter 1, but is more convenient for our current purposes. Let $\vec{K}_n$ be the complete digraph on n vertices. Thus, $\vec{K}_n$ is a digraph with n vertices, labeled say $1, \ldots, n$, and all $n(n-1)$ possible arcs (i, j) for $1 \leq i, j \leq n$, $i \neq j$. Given a weight function w defined on the arcs of $\vec{K}_n$, the TSP may be defined as the problem of

finding a dicircuit including all vertices and having minimum total weight. As in Example 1.1.2, we call a dicircuit including all vertices a *tour*.

Let us try to formulate the TSP as an LP. We begin with the following *integer linear program* (ILP):

$$\begin{array}{lll} \min & w^T x & \\ \text{s.t.} & \sum_{j \neq k} x_{jk} = 1 & (k = 1, \ldots, n) \\ & \sum_{k \neq j} x_{jk} = 1 & (j = 1, \ldots, n) \\ & x_{jk} \in \{0, 1\} & (\text{all } j, k) \end{array} \tag{4.4}$$

where the x_{jk} have the interpretation of selecting or not selecting the arc (j, k) when $x_{jk} = 1$ or 0, respectively. The first n constraints say that a tour must enter each vertex exactly once, and the second n constraints that a tour must leave each vertex exactly once. (4.4) is called an ILP because it is an LP except for the restriction that all variables take on integral values.

		Vertex				
		1	2	3	4	5
	1	–	3	5	2	7
	2	3	–	6	4	1
Vertex	3	5	6	–	2	8
	4	2	4	2	–	3
	5	7	1	8	3	–

$\vec{K}_5$ arc weights

Figure 4.1: The entry in row i and column j specifies the weight, $w(i, j)$, of arc (i, j).

Example 4.2.1 Consider the symmetric weighting of $\vec{K}_5$ given in Figure 4.1. ILP (4.4) then has 20 variables and 10 constraints, each involving 4 variables:

$$\begin{aligned}
\min \quad & 3x_{12} + 5x_{13} + 2x_{14} + 7x_{15} + 6x_{23} + \\
& 4x_{24} + x_{25} + 3x_{21} + 2x_{34} + 8x_{35} + \\
& 5x_{31} + 6x_{32} + 3x_{45} + 2x_{41} + 4x_{42} + \\
& 2x_{43} + 7x_{51} + x_{52} + 8x_{53} + 3x_{54}
\end{aligned}$$

$$\begin{aligned}
\text{s.t.} \quad & x_{21} + x_{31} + x_{41} + x_{51} = 1 \\
& x_{12} + x_{32} + x_{42} + x_{52} = 1 \\
& x_{13} + x_{23} + x_{43} + x_{53} = 1 \\
& x_{14} + x_{24} + x_{34} + x_{54} = 1 \\
& x_{15} + x_{25} + x_{35} + x_{45} = 1 \\
& \\
& x_{12} + x_{13} + x_{14} + x_{15} = 1 \\
& x_{23} + x_{24} + x_{25} + x_{21} = 1 \\
& x_{34} + x_{35} + x_{31} + x_{32} = 1 \\
& x_{45} + x_{41} + x_{42} + x_{43} = 1 \\
& x_{51} + x_{52} + x_{53} + x_{54} = 1 \\
& \\
& x_{jk} = 0 \text{ or } 1 \text{ (all } j, k) \qquad \square
\end{aligned} \tag{4.5}$$

The ILP (4.4) is not a correct formulation of the TSP because, while it does guarantee that all vertices are visited, it allows for a solution made up of several *subtours*, smaller dicircuits that include only a subset of the vertices (see the continuation of Example 4.2.1 below). Nevertheless, (4.4) does have one very nice property. Even though it is formulated as an ILP, rather than an LP, and integer programming is, in general, $\mathcal{NP}$-hard, in this case that integrality restrictions are redundant: We can replace (4.4) by its *LP relaxation*, that is, replace each of the integrality restrictions $x_{jk} \in \{0, 1\}$ by $0 \leq x_{jk} \leq 1$, and the answer does not change. We shall prove this shortly.

Example 4.2.2 ((4.2.1) continued) The solution of the LP relaxation of (4.5) is $x_{13} = x_{25} = x_{34} = x_{41} = x_{52} = 1$, with all other variables equal to 0. This solution has total weight 11, and, as claimed above, is indeed integral. Note however that it is made up of two subtours: $(1, 3, 4, 1)$ and $(2, 5, 2)$. $\square$

Before attacking the subtour elimination problem, we prove the above claimed integrality property of the (4.4) formulation.

Proposition 4.2.3 *The LP relaxation of (4.4) has an integral optimal solution.*

Proof. Let (4.4′) denote the LP relaxation of (4.4). Let $Ax = b$ be the system of equality constraints in (4.4). We show that A is *totally unimodular*, that is, that

the determinant of every square submatrix of A has value ± 1 or 0. To see that this will prove the proposition, note that the set of feasible solutions of (4.4′) is clearly nonempty and bounded (bounded because each variable is individually bounded, and feasible because any tour is a feasible solution). Hence, by Proposition 4.1.5, the associated LP has an optimal solution that is an extreme point. But by Proposition 4.1.3, any extreme point of the feasible region can be obtained by setting some of the x_{jk} to 0 or 1, and solving the resulting subsystem of $Ax = b$ uniquely for the remaining variables. Cramer's rule together with the total unimodularity of A implies that this solution must be integral.

To prove the total unimodularity claim, assume it is false and let B be a minimal square submatrix of A with determinant other than ± 1 or 0. Note that the rows of A partition into two sets such that every column in each set contains exactly one 1, and otherwise 0s. This partition induces a partition of the rows of B such that each column has *at most* one 1 in each set. If, in fact, every column has exactly one 1 in each set, then summing the rows in each part of the partition we obtain a vector of all 1s. Hence, B is singular and has determinant 0, a contradiction.

Similarly, there can be no column of all 0s. Hence, there is a column with exactly one 1. But then expanding the determinant of B on this column, we obtain $\det B = \pm \det B'$, where B' is a proper submatrix of B. This is a contradiction, since by the minimality of B, $\det B'$ is ± 1 or 0. □

The idea of the above proof, that of using total unimodularity to show that a particular ILP can be solved as an LP is due to A. Hoffman and J. B. Kruskal (1956) ["Integral boundary points of convex polyhedra," in *Linear Inequalities and Related Systems*, H. W. Kuhn and A. W. Tucker, eds., Princeton Univ. Press, Princeton, N. J., 223–246].

Let us now turn to the problem of eliminating subtours. C. E. Miller, A. W. Tucker and R. A. Zanlin (1960) ["Integer programming formulations and traveling salesman problems," *Journal of the Association of Computing Machinery* **7** 326–329] proposed the following clever formulation:

$$\begin{array}{lll} \min & w^T x & \\ s.t. & \sum_{k \neq j} x_{jk} = 1 & (j = 1, \ldots, n) \\ & \sum_{j \neq k} x_{jk} = 1 & (k = 1, \ldots, n) \\ & u_j - u_k + n x_{jk} \leq n - 1 & (2 \leq j, k \leq n, j \neq k) \\ & x_{jk} \in \{0, 1\} & (\text{all } j, k, j \neq k) \end{array} \tag{4.6}$$

We leave it to the reader to verify that this formulation does indeed eliminate subtours, and is thus a correct ILP formulation of the TSP. This would seem to imply that we have in fact found a acceptable formulation of the TSP. However, there is a difficulty. While (4.6) is a correct ILP formulation of the TSP, it is also a more typical ILP than (4.4) in that it does not have the integrality property proved in (4.2.3). Obviously the new constraint matrix is not totally unimodular (it doesn't

even have entries restricted to $\{0, \pm 1\}$), and so certainly the idea we applied to (4.4) cannot be applied.

Example 4.2.4 ((4.2.1) continued) (4.6) adds 12 constraints to (4.5). An optimal solution to the associated LP relaxation is $x_{13} = 1$, $x_{25} = 0.6$, $x_{21} = 0.4$, $x_{34} = 1$, $x_{45} = 0.4$, $x_{41} = 0.6$, $x_{52} = 1$, $u_2 = 1.0$, $u_4 = 2.0$, which has total weight 12.2. □

In an attempt to more clearly understand the difficulties that arise in (4.4) and (4.6), we consider a "polyhedral approach." To this end, let A be the set of arcs of $\vec{K}_n$. For $I \subseteq A$ define the *incidence* or *characteristic vector* of I, x^I, by $x^I(a) = 1$ if $a \in I$ and $x^I(a) = 0$ if $a \in A \backslash I$. Let $\mathcal{I}$ be the family of subsets of arcs of A that form tours. Let

$$P_{TSP} = \text{conv}\{x^I : I \in \mathcal{I}\}.$$

Then it is easy to see that solving $\min\{w^T x : x \in P_{TSP}\}$ solves the TSP on $\vec{K}_n$ for a given weighting w: If x^* is an optimal solution of this optimization problem, then $x^* \in P_{TSP}$ implies x^* is a convex combination of a finite set of incidence vectors of tours, $x^* = \sum_{j \in J} \alpha_j x^{I_j}$. If $w^T x^* < w^T x^{I_j}$ $(j \in J)$, then clearly

$$w^T x^* = \sum_{j \in J} \alpha_j w^T x^{I_j} > \sum_{j \in J} \alpha_j w^T x^* = w^T x^*,$$

a contradiction. Hence, some I_j $(j \in J)$ is optimal.

Denote the feasible regions of the LP relaxations of (4.4) and (4.6) by P_a and P_b, respectively. Clearly, both P_a and P_b contain P_{TSP}; the difficulty is that both properly contain P_{TSP}, and that they are not good enough approximations. The following formulation provides a better approximation. It was suggested in a seminal paper by G. B. Dantzig, D. R. Fulkerson and S. M. Johnson (1954) ["Solution of a large scale traveling salesman problem," *Operations Research* **2** 393–410]:

$$\begin{array}{lll} \min & w^T x & \\ \text{s.t.} & \sum_{k \neq j} x_{jk} = 1 & (j = 1, \ldots, n) \\ & \sum_{j \neq k} x_{jk} = 1 & (k = 1, \ldots, n) \\ & x(\delta^-(X)) \geq 1 & (\emptyset \subset X \subset V) \\ & x_{jk} \in \{0, 1\} & (\text{all } j, k, j \neq k) \end{array} \tag{4.7}$$

Note that the constraints $x(\delta^-(X)) \geq 1$ do eliminate subtours since if x is integral and X is the vertex-set of some subtour determined by x, then $x(\delta^-(X)) = 0$.

At first (4.7) would seem to be less useful than (4.6), since the number of subtour elimination constraints is now expenential, $2^n - 2$. For any reasonably large problem, (4.7) cannot even be written down. However, in their paper Dantzig, Fulkerson and Johnson found that they could "generate" these constraints as they needed them by using a network-flow technique. In this way they were actually able to solve *by hand* the 48-city TSP ($n = 48$) corresponding to traveling through all the state capitals in the continental US.

Example 4.2.5 ((4.2.1) continued) Formulation (4.7) adds $2^5 - 2 = 30$ subtour elimination constraints to (4.6). The optimal solution to the LP relaxation is $x_{12} = x_{25} = x_{31} = x_{43} = x_{54} = 1$ with total weight 14. Since the solution is integral, and contains no subtours, it must be optimal for the original TSP. Indeed, a more detailed examination of the associated LP solution, and in particular the optimal values of the dual variables (not shown here, but provided by any standard solution technique) reveals even more information: Most of the dual variables corresponding to subtour elimination constraints are 0, and hence these constraints are not binding on the solution—they can be deleted. Indeed, only one of the additional 30 constraints is needed to produce the desired optimal solution, $X = \{1, 3, 4\}$:

$$x_{12} + x_{15} + x_{35} + x_{32} + x_{45} + x_{42} \geq 1 \tag{4.8}$$

Note that this constraint explicitly eliminates the subtours $(1, 3, 4, 1)$ and $(2, 5, 2)$ found in solving (4.5). Thus, we could have found it by first solving (4.5) and then examining the solution.

If one writes out the dual of (4.5) with (4.8) appended, it is readily checked that $y^T = [6\ 1\ 8\ 3\ 4\ -3\ -3\ -1\ -6\ 0\ 5]$ is feasible. Note in particular that y has 11 components, the first 10 corresponding to the constraints of (4.5) and the last corresponding to (4.8). It also has some negative components, permissible since the LP primal has equality constraints corresponding to these components. Finally, note that the sum of the coordinates of y is 14. This proves, using weak duality, that 14 is a minimum-weight tour. □

4.3 An Exact Defining System for P_{TSP}?

The previous section is meant to illustrate how polyhedral and techniques might be used to solve hard combinatorial problems. Contained in the section are the beginnings of what has proved to be a very successful "deep-cut method" for the solution of combinatorial problems. Apart from its purely illustrative value, there are also several important theoretical ideas contained in this development. First, the idea of finding a good approximation to the convex hull of the desired combinatorial solutions—in the case of the TSP, a good approximation to the polyhedron P_{TSP}; the idea of being able to generate this approximation as it is needed, without having to explicitly write down all constraints (the way in which (4.8) could be found from the solution to (4.5)); finally, the idea that when this approach succeeds, it not only provides a solution, but, via LP weak duality, a proof of optimality. The two last points came up first when considering the formulation (4.7).

Let us now consider some of these issues in a more general context. It follows from the Farkas-Minkowski-Weyl Theorem, (4.1.4), that there does *exist* an exact description of P_{TSP} in the form $\{x : Ax \leq b\}$, but can we actually find A and b in

practice? More precisely, can we "find" an A and b that describe P_{TSP} and are not too complicated? For example, we certainly don't want A to contain coefficients with too many significant digits, so that even specifying an entry is in itself a laborious procedure.

Considerations in theoretical computer science seem to suggest that the problem of finding such an A will be difficult, for this would imply that the TSP has a good proof of optimality, and hence is in the class $\mathcal{NP} \cap$ co-$\mathcal{NP}$. To see this we apply LP duality theory, as described below

Consider the LP $\min\{w^T x : Ax \leq b, x \geq 0\}$, where $P_{TSP} = \{x : Ax \leq b\}$; the added nonnegativity conditions $x \geq 0$ clearly do not change P_{TSP}. The dual of this problem has constraints $A^T y \geq w$, $y \geq 0$. Now the primal clearly has an optimal solution since the incidence vector of any optimal tour is also optimal for the LP. Let x^* be the incidence vector of such an optimal tour. Then by strong duality, (4.1.2), there is an optimal dual solution y^*, and $w^T x^* = b^T y^*$. Now to "prove optimality" we apply weak duality; thus, we must show how it can be quickly checked, in polynomial time, that $Ax^* \leq b$, $x^* \geq 0$, $A^T y* \geq w$, and $y^* \geq 0$. The matrix A is likely, in fact, certain, to be huge, having exponentially many rows. Hence, it is not a priori clear how this checking is to be done. However, as x^* is the incidence vector of a tour, $x^* \geq 0$ is evident, and $Ax^* \leq b$ we can check implicitly. After all, if we have a theorem that claims $P_{TSP} = \{x : Ax \leq b\}$, and in particular that P_{TSP} is a subset of this latter polyhedron, then it suffices simply to show that x^* is the incidence vector of a tour! This takes time $O(n)$.

Finally, consider y^*. Here the problem is somewhat more difficult. Because A has a large number of rows, y^* could in principal have a large number nonzero components, and thus, be itself non-polynomial in size. However, this difficulty can be avoided, for we can assume that y^* has been chosen to be an extreme point of the dual feasible region, $\{y \geq 0 : A^T y \leq w\}$. This follows from Proposition 4.1.5 since $y \geq 0$ prevents this set from having a nontrivial lineality space. Suppose that A is $K \times n$, where, as we have noted, K can be large. Then by Proposition 4.1.3, y^* is the unique solution of some subsystem $(A')^T y = w'$, $y_j = 0$ $(j \in J)$. Since y has K components, this system must include at least K equations, or the rank of the system cannot be large enough to force a unique solution. But if the system has at least K equations, then $|J| \geq K - n$, since A has n columns. Hence, y has at most n positive components. It follows that checking $y^* \geq 0$ is straightforward, as is checking $A^T y^* \leq w$, the latter computation taking time at most $O(n^2)$. Finally, we need to verify $w^T x^* = b^T y^*$, which is trivial (given that we know y^* does not have too many nonzero coordinates), and the proof is complete.

The above arguments suggest that a good, exact description of P_{TSP} is unlikely to be found. This is a negative result. However, the practical implications of the above ideas, particularly the examples of the previous section, are meant to be positive. They illustrate that even though a complete understanding of P_{TSP} is

desired, it is not necessarily needed to get optimality. Indeed, we have proved that there *are* always a small number of constraints that suffice, for a given w, to get an optimal solution and prove its optimality. What we have not shown is how to find these constraints, and that the constraints themselves are necessarily easy to write down.

Exercises

4.1 Show that the formulation of the TSP given by (4.6) is valid. That is, show that an integral x satisfies the constraints of (4.6) only iff x is the incidence vector of a tour.

4.2 The *recession cone* of a convex set P, rec P, is defined by rec $P = \{y : y+x \in P \, \forall x \in P\}$. Show that a closed convex set P is unbounded iff rec P contains some nonzero vector. (Can you give an example showing that this is false if P is not closed?)

I will accept a proof that this is true for polyhedra, which is simpler. A proof for general closed convex sets seems to be an exercise in analysis.

4.3 Show that $\dim P_{TSP} = n^2 - 3n + 1$. (Hint: The first step is to use the constraints of (4.2.3) to show that $\dim P_{TSP} \leq n^2 - 3n + 1$.)

Chapter 5

Facets of Polyhedra

5.1 Introduction

In this chapter we study the polyhedron of the minimum spanning tree problem and give a complete description. A key idea is the notion of a "facet," or "maximal face." For hard combinatorial problems, where finding a complete description of the underlying polyhedra is expected to be hard (see the last section of Chapter 4), finding theoretical descriptions of facets is a crucial ingredient in developing good computational procedures. In the context of integer programming, facets are sometimes called *deep* or *strong* cuts.

In Chapter 6 we discuss the ellipsoid method and prove a result of Grötschel, Lovász and Schrijver showing that polynomial-time optimization is equivalent to the existence of a polynomial-time algorithm for finding separating hyperplanes, given the ellipsoid method. This fact yields the only known polynomial-time algorithms for several important combinatorial problems. It also provides a general theoretical basis for the polyhedral approach to CO problems: It implies that if a complete description of a polyhedron can be found, *and* the separation problem can be solved, then optimization is possible in polynomial time over that polyhedron.

5.2 More Polyhedral Preliminaries

We need a notion of dimension for polyhedra, from which come the tools to deal with facets.

An *affine combination* of vectors $x^j \in \mathbf{R}^n$ $(j = 1, \ldots, n)$ is a linear combination of the form $\alpha_1 x^1 + \ldots + \alpha_k x^k$, where $\alpha_1 + \ldots + \alpha_k = 1$. Thus, an affine combination is a convex combination in which some coefficients may be negative. For $X \subseteq \mathbf{R}^n$, the *affine hull* of X, aff X, is defined to be the set of all finite affine combinations

of vectors in X. X is said to be *affine* if aff $X = X$. For $a \in \mathbf{R}^n$, $X + a$ denotes the set $\{x + a : x \in X\}$.

Proposition 5.2.1 *X is affine iff $X = S + a$ for some linear subspace S and some vector a; indeed, if X is affine then $X - a$ is a linear subspace, the same linear subspace for any $a \in X$.*

Proof. Suppose $X = S + a$, where S is a linear subspace. Then an affine combination $x = \sum_{j \in J} \alpha_j x^j$ of vectors in X satisfies $x = a + \sum_{j \in J} \alpha_j (x^j - a) \in X$, since $\sum \alpha_j = 1$, $x^j - a \in S$ $(j \in J)$ and S is closed under linear combinations.

To prove the converse, let X be affine and let $a \in X$. We show that $S = X - a$ is a subspace. For a linear combination $s = \sum_{j \in J} \alpha_j (x^j - a)$ of vectors in $X - a$, we have

$$s = \sum_{j \in J} \alpha_j x^j + (1 - \sum_{j \in J} \alpha_j) a - a \in X - a$$

since $\sum \alpha_j + (1 - \sum \alpha_j) = 1$, $a \in X$ and X is closed under affine combinations. □

If X is a convex set, we define the *dimension* of X, $\dim X$, to be the dimension of the subspace $(\text{aff } X) - a$, where $a \in X$.

Now, let $P = \{x : Ax \leq b\}$ be a polyhedron. Let $A^= x \leq b^=$ be the subsystem of $Ax \leq b$ such that $A^= x = b^=$ for all $x \in P$. The equations $A^= x = b^=$ are called *implicit equations* of P. The remaining inequalities in the system $Ax \leq b$ are denoted $A^+ x \leq b^+$. Note that if $P \neq \emptyset$, then $A^+ x < b^+$ for some $x \in P$ (if $A = A^=$ we take this to be vacuously true, and otherwise take a proper convex combination of a set of vectors such that for each of the inequalities in $A^+ x \leq b^+$ one of the vectors in the set satisfies this inequality strictly). The next proposition implies that $\{x : A^= x = b^=\}$ is independent of the choice of A and b, depending only on P.

Proposition 5.2.2 *If $P \neq \emptyset$, then* aff $P = \{x : A^= x = b^=\}$.

Proof. Suppose $z = \alpha_1 z^1 + \ldots + \alpha_k z^k$, where $z^j \in P$ $(j = 1, \ldots, k)$ and $\alpha_1 + \ldots + \alpha_k = 1$. Then by the definition of $A^=, b^=$ we have $A^= z^j = b^=$ for each j, and so

$$A^= z = \sum_{j=1}^{k} \alpha_j A^= z^j = \sum_{j=1}^{k} \alpha_j b^= = b^=.$$

This proves aff $P \subseteq \{x : A^= x = b^=\}$.

To prove the converse, let $z \in \{x : A^= x = b^=\}$, and let $y \in P$ be such that $A^+ y < b^+$. Now for $\alpha > 0$, define $y^\alpha = y + \alpha(z - y)$. Note that

$$A^= y^\alpha = A^= y + \alpha(A^= z - A^= y) = b^= + \alpha(b^= - b^=) = b^=;$$

moreover, for the remaining inequalities in $Ax \leq b$ we have $A^+y^\alpha = A^+y + \alpha(A^+z - A^+y)$. Hence, $A^+y < b^+$ implies that for sufficiently small $\alpha > 0$, $A^+y^\alpha < b^+$, and so $y^\alpha \in P$. But $z = \frac{1}{\alpha}y^\alpha + (1 - \frac{1}{\alpha})y$, and so $z \in \text{aff}\, P$. □

The above result implies that $\{x : A^=x = b^=\}$ is independent of the choice of A, b. It also implies:

Corollary 5.2.3 $\dim P = n - \text{rank}\, A^=$. □

We next define *face* and *facet*. A face of a polyhedron P is a nonempty set of the form $F = \{x \in P : a^Tx = \beta\}$ where a^Tx must be a ***valid inequality*** for P, that is, $P \subseteq \{x : a^Tx \leq \beta\}$. The hyperplane $\{x : a^Tx = \beta\}$ is then called a ***supporting hyperplane*** of P. A face F of P is ***proper*** if $F \neq P$. A maximal proper face is a facet.

We have the following relationship between the faces of a polyhedron and any representing set of inequalities.

Proposition 5.2.4 *F is a face of a polyhedron $P = \{x : Ax \leq b\}$ iff $F \neq \emptyset$ and $F = \{x \in P : A'x = b'\}$ for some subsystem $A'x \leq b'$ of $Ax \leq b$.*

Proof. For one direction of the proof let $F = \{x \in P : A'x = b'\}$. Define $a = e^TA'$ and $\beta = e^Tb'$, where e is a vector of all ones. Then clearly $\{x \in P : a^Tx = \beta\} = F$, since $x \in P$ (that is, $Ax \leq b$) implies $e^TA'x = e^Tb'$ iff $A'x = b'$. Hence F is a face.

The proof of the converse is a bit more tedious. Let $F = \{x \in P : a^Tx = \beta\}$ be a face, and let $A'x \leq b'$ be the set of all inequalities of $Ax \leq b$ satisfied at equality by every $x \in F$. Let $F' = \{x \in P : A'x = b'\}$. Clearly $F \subseteq F'$. Suppose there is a vector $y \in F' \backslash F$. Let $A''x \leq b''$ be the set of inequalities from $Ax \leq b$ not included in $A'x \leq b'$. We may assume that there is a $z \in F$ such that $A''z < b''$. This follows from the convexity of F and the choice of A' and b'. Let $z^\alpha = z + \alpha(y - z)$. Since $A''z < b''$ and $A'z = b'$, it follows that $z^\alpha \in P$ for all α sufficiently near zero. But $a^T(y - z) \neq 0$, and so $a^Tz^\alpha > \beta$ for some such α, contradicting the fact that $a^Tx \leq \beta$ is valid. This contradiction completes the proof. □

The above result implies that P has only a finite number of faces. The next result characterizes facets in terms of a defining system of inequalities. Call a defining system *irredundant* if every inequality is *essential*, that is, if the removal of any inequality from the system enlarges the polyhedron.

Theorem 5.2.5 *Let $Ax \leq b$ be an irredundant defining system for a polyhedron P. Then F is a facet of P iff $F = \{x \in P : a^Tx = \beta\}$ for some row $[a^T\ \beta]$ of $[A^+\ b^+]$.*

Proof. To prove one half of the theorem, let $[a^T\ \beta]$ be a row of $[A^+\ b^+]$ and let $F = \{x \in P : a^Tx = \beta\}$. Clearly F is a proper face since none of the inequalities

in $A^+x \leq b^+$ is implicit. To show that F is maximal, let $x^1 \in P$ be such that $A^+x^1 < b^+$. By irredundancy there is an x^2 such that $A^=x^2 = b^=$, $A'x^2 \leq b'$ and $a^Tx^2 > \beta$, where $A'x \leq b'$ is the system $A^+x \leq b^+$ with $a^Tx \leq \beta$ removed. But then for some convex combination x^3 of x^1 and x^2 we have $A^=x^3 = b^=$, $A'x^3 < b'$ and $a^Tx^3 = \beta$, which implies that F is maximal. In particular, by (5.2.4), if there were a proper face containing F it would be given by some of the inequalities from $Ax \leq b$, and we have just shown that every such inequality that is not implicit, and is different from $a^Tx \leq \beta$, excludes some point of F, namely x^3.

For the converse let F be a facet. Then by (5.2.4), $F = \{x \in P : A'x = b'\}$ for some subsystem of $A^+x \leq b^+$. Taking one of the equalities from $A'x = b'$ suffices, since none are implicit. □

Corollary 5.2.6 *If F is a face of P, then F is a facet iff* $\dim F = \dim P - 1$.

Proof. Clearly two affine sets, one containing the other and both with the same dimension, are equal, because subspaces of equal dimension are equal; moreover, if F_1 and F_2 are two faces of the same polyhedron, then $F_1 = F_2$ iff $\operatorname{aff} F_1 = \operatorname{aff} F_2$. Hence, $\dim F = \dim P - 1$ for a face F, implies F is a facet.

To prove the converse let $F = \{x \in P : a^Tx = \beta\}$ where $[a^T \ \beta]$ is a row of $[A \ b]$ and $Ax \leq b$ is an irredundant system determining P—we may assume F has this form by (5.2.5). Now clearly $F = \{x : Ax \leq b, a^Tx \geq \beta\}$, and by the proof of (5.2.5), the only implicit equalities in this system are $A^=x = b^=$ and $a^Tx = \beta$ (x^3 from the proof of (5.2.5) shows this). Hence, by Corollary 5.2.3, $\dim F = n - \operatorname{rank} A^= - 1 = \dim P - 1$. □

Corollary 5.2.7 *Suppose F is a facet of $P = \{x : Ax \leq b\}$ and*

$$F = \{x \in P : \underline{a}^Tx = \underline{\beta}\} = \{x \in P : \overline{a}^Tx = \overline{\beta}\}$$

where $\underline{a}^Tx \leq \underline{\beta}$ and $\overline{a}^Tx \leq \overline{\beta}$ are valid. Then there exists a vector z and a scalar $\alpha > 0$ such that $[\underline{a}^T \ \underline{\beta}] = \alpha[\overline{a}^T \ \overline{\beta}] + z^T[A^= \ b^=]$.

Proof. Let $[\underline{A} \ \underline{b}]$ and $[\overline{A} \ \overline{b}]$ be $[A^= \ b^=]$ with $[\underline{a}^T \ \underline{\beta}]$ and $[\overline{a}^T \ \overline{\beta}]$, respectively, appended as the last row. Then $\dim F = \dim P - 1$ implies $\operatorname{aff} F = \{x : \underline{A}x = \underline{b}\} = \{x : \overline{A}x = \overline{b}\}$, since both of the latter sets contain F, both are affine, and both have dimension less than $\dim\{x : A^=x = b^=\} = \dim P$. This proves that $[\underline{a} \ \underline{\beta}]$ is represented as claimed in the corollary, where $\alpha \neq 0$. To prove $\alpha > 0$, let $x \in P$ be such that $\overline{a}^Tx < \overline{\beta}$. Then $\alpha < 0$ implies $\underline{a}^Tx = \alpha\overline{a}^Tx + z^TA^=x > \alpha\overline{\beta} + z^Tb^= = \underline{\beta}$, a contradiction. □

5.3 A Minimum-Spanning-Tree Polyhedron

We saw at the end of the previous chapter that finding a complete description of the polytope corresponding to the traveling salesman problem is likely to be very difficult. In this section we show that, on the contrary, such a result *can* be obtained for the polyhedron associated with the minimum spanning tree problem.

Let G be an undirected graph and let $\mathcal{I}$ be the family of edge-sets of forests of G. Define $P_{MST} = \text{conv}\{x^I : I \in \mathcal{I}\}$. We call P_{MST} the *minimum-spanning-tree, MST, polyhedron.* The following theorem gives a complete description of P_{MST}. The proof is derived from a proof for the polyhedron of "independent sets of a matroid" given in [5].

Theorem 5.3.1 *Let $G = (V, E)$ be a graph and let $E_0 = \{e \in E : e \text{ is a loop}\}$. Then P_{MST} for G is the set of solutions of the following system of inequalities:*

$$\begin{array}{lll} (i) & x(E(W)) \leq |W| - 1 & \forall W \subseteq V, |W| \geq 2; \\ (ii) & x(e) \geq 0 & \forall e \in E; \\ (iii) & x(e) = 0 & \forall e \in E_0. \end{array}$$

Remark: $E(W)$ denotes the set of edges in G with both ends in W. Similarly, for a subset of edges A of G, $V(A)$ denotes the set of vertices incident to A. Note that (i) trivially holds if $|W| = 1$.

Proof. Let P denote the solution set of (i,ii,iii). First we show $P_{MST} \subseteq P$. Since P is convex, it suffices to show that x^I satisfies (i,ii,iii) for each edge-set I of a forest. Obviously (ii) holds, and (iii) holds because no loop can be in any forest. To see that (i) holds let $W \subseteq V(G)$ and let $W_1, \ldots, W_k$ be the vertex-sets of the connected components of $I \cap E(W)$. Then we have

$$\begin{array}{rcl} x^I(E(W)) & = & |I \cap E(W)| \\ & = & |I \cap E(W_1)| + \ldots + |I \cap E(W_k)| \\ & = & |W_1| - 1 + \ldots + |W_k| - 1 \\ & \leq & |W| - 1, \end{array}$$

where the third equality follows because $I \cap E(W_j)$ is a tree ($j = 1, \ldots, k$) (see 2.1.1). This proves $P_{MST} \subseteq P$.

To show the converse, $P \subseteq P_{MST}$, we show two things: (a) that $\text{aff}\, P_{MST} = \{x : x(e) = 0, e \in E_0\}$, and (b) that any facet of P_{MST} is a nonnegative multiple of some inequality of type (i) or (ii) plus a linear combination of equations of type (iii). Since by the Farkas-Minkowski-Weyl Theorem, (4.1.4), P_{MST} is the solution set of some system of inequalities, and hence of an irredundant system (in the sense of the previous section), it follows from Theorem 5.2.5 and the above claims (a) and

(b), once they have been proved, that any $x \in P$ must satisfy all the inequalities of any such irredundant defining system, and hence that $x \in P_{MST}$.

First we prove (a). The equations (iii) are valid for P_{MST} (as noted in the first paragraph of the proof) and are linearly independent. Hence, $\dim P_{MST} \leq |E| - |E_0|$. But P_{MST} contains the 0-vector and the linearly independent vectors $\{x^{\{e\}} : e \in E \backslash E_0\}$, and so $\dim P_{MST} \geq |E| - |E_0|$. Hence, equality holds. This proves (a).

To prove (b), let $F_a = \{x \in P : a^T x = \beta\}$ be a facet of P_{MST} and let $\mathcal{I}_a = \{I \in \mathcal{I} : x^I \in F_a\}$. We may assume that $a_j = 0$ for $j \in E_0$ since scalar multiples of the valid equations $x(e) = 0$ $(e \in E_0)$ may be subtracted from $a^T x = \beta$ without changing F_a. There are now two cases to consider:

Case 1. Suppose $a_j < 0$ for some j. Now if $j \in I$ for some $I \in \mathcal{I}_a$, then $\beta \geq a^T x^{I \backslash \{j\}} = a^T x^I - a_j > \beta$, a contradiction. It follows that $x \in F_a$ implies $x_j = 0$, since by the definition of P_{MST}, $F_a = \text{conv}\{x^I : I \in \mathcal{I}_a\}$. Hence $F_a \subseteq F_j = \{x \in P : x_j = 0\}$. But F_j is a proper face of P_{MST} since $j \in E \backslash E_0$. Hence, by the maximality of facets, $F_a = F_j$. Since $a_k = 0$ for $k \in E_0$, Corollary 5.2.7 implies that $a^T x \leq \beta$ is a positive multiple of the inequality $-x_j \leq 0$.

Case 2. Suppose $a \geq 0$ and let $A = \{j : a_j > 0\}$. We claim that every $I \in \mathcal{I}_a$ is the edge-set of a maximal forest in A. Suppose not. Let $I \in \mathcal{I}_a$ be such that $|I| < m$, where m is the size of a maximal forest in A. Extend I to a maximal forest $I \cup K$ of A. Then we have

$$\beta \geq a^T x^{I \cup K} > a^T x^I = \beta,$$

a contradiction. Hence F_a is a subset of the face $F = \{x \in P : x(A) = m\}$; moreover, F is a proper face since A is a nonempty subset of $E \backslash E_0$. It follows that $F_a = F$, and so $a_k = 0$ for $k \in E_0$ implies that $a^T x \leq \beta$ is positive multiple of the inequality $x(A) \leq m$.

Now to complete the proof of this case, let W be the vertex-set of a component of $G(A)$ and let $F' = \{x : x(E(W)) = |W| - 1\}$. F' is a proper face of P and $F_a \subseteq F'$. As above we conclude that $F_a = F'$ and that $A = E(W)$. □

The previous result gives a defining system for P_{MST}, but, as the last part of the proof suggests, it is not an irredundant system. Define a graph to be 2-*connected* if every pair of edges is contained is a common circuit. If the subgraph induced by $E(W)$ is not 2-connected for some subset of vertices W, then where $W_1, \ldots, W_k$ are the vertex-sets of the "2-connected components" of the subgraph on $E(W)$, (iii) for W is implied by (iii) for W_j $(j = 1, \ldots, k)$, an observation that follows from the

observation that a forest of a graph is maximal iff it is a spanning tree of each of the 2-connected components of the graph. This is the only redundancy that occurs.

Theorem 5.3.2 *If $W \subseteq V(G)$ and $E(W)$ is a 2-connected subset of edges, then the inequality $x((E(W)) \leq |W| - 1$ defines a facet of P_{MST}.* □

Exercises

5.1 Show that exact Gaussian elimination for rational linear systems is polynomial.

Remark: In order to make the intent of the problem clear, we first specify what is meant by Gaussian elimination, and why it is not obviously polynomial. Let $Ax = b$ be an $n \times n$ system of equations involving only rational coefficients, and assume for convenience that A is nonsingular. Suppose that $a_{11} \neq 0$. Then eliminating a_{11} means replacing a_{ij} and b_i for $i \geq 2$ by $a_{11}a_{ij} - a_{i1}a_{1j}$ and $a_{11}b_i - a_{i1}b_1$, respectively. Repeating this elimination procedure $n - 1$ times, following each elimination by a possible reordering of rows to insure that the "pivot element" is nonzero, is what we call Gaussian elimination. Clearly, once the procedure is completed, solving for x is merely a matter of back substitution.

How efficient is this procedure? The number of arithmetic operations is clearly polynomial, $O(n^3)$, but how much work does each of the operations take? Since we have said that the arithmetic must be exact, this work cannot be ignored. Each of the numbers in A and b is rational, and so can be taken as represented as a ratio of two integers. The size of each of the numbers is thus the total number of digits in the numerator and denominator (this is, after all, to within a constant how much space it takes to store these integers). Now if M is the size of the largest of the a_{ij}, then after one elimination, the size of the largest coefficient can apparently be as big as $O(M^2)$. Thus, the number of digits may roughly double. (Note that simply normalizing does not help because of our insistence on exact arithmetic. Indeed, this would seem to make the growth worse.) After $n - 1$ eliminations, the number of digits becomes $O(2^{n-1})$, an this is not polynomial!

Jack Edmonds was the first to prove that there is a way around this difficulty. He observed that after the second elimination is carried out, a_{11} divides every entry in rows 3 up to n. One can prove the desired result by verifying Edmonds' observation, and showing how its repeated application can be used to keep the number of digits from growing too rapidly.

5.2 Prove Theorem 5.3.2: If W is a subset of vertices of a graph G and $E(W)$ is a 2-connected subset of edges, then the inequality

$$x(E(W)) \leq |W| - 1$$

defines a facet of P_{MST}. (A graph is 2-connected if every pair of edges is contained in a circuit.)

Outline of Proof: It is sufficient to prove the result when G has no loops. Then P_{MST} has dimension $|E|$. Let $\mathcal{I}$ be the family of all edge-sets of forests I such that x^I satisfies $x^I(E(W)) = |W| - 1$. Let A be a matrix the rows of which are the incidence vectors x^I ($I \in \mathcal{I}$). Conclude that if the given inequality is not a facet, then $\text{rank}(A) \leq |E| - 1$. Show that for any maximal forest I of $E(W)$ and any element $f \in E \backslash E(W)$, $x^I \cup \{f\}$ is a row of A.

Now, let $z \in \mathbf{R}^E$ be a nonzero vector such that $Az = 0$ (A has $|E|$ columns). Note that z is zero outside $E(W)$. Let $I_1 = \{f \in E(W) : z_f < 0\}$ and $I_2 = \{f \in E(W) : z_f \geq 0\}$. Both sets are nonempty (A has no zero column). Show that no circuit in $E(W)$ intersects both I_1 and I_2, a contradiction. $\square$

Chapter 6

Ellipsoids

6.1 Overview

We present a restricted version of the ellipsoid method, Algorithm 6.3.7, and prove it is polynomial time bounded. Then we present a result of Grötschel Lovász and Schrijver proving that optimization is equivalent to separation. This latter result has important implications for combinatorial optimization. Our development is based on Chapter 13 of [10], and in part on [4], pp. 443-454.

The ellipsoid method was developed over a period of years by several Russian mathematicians as a way to solve general nonlinear programs, and convex programs in particular. The method can be viewed as having emerged from two separate lines of development.

In 1964 N. Z. Shor presented a general "subgradient method," a generalization of what has come to be known as a "relaxation method." In this method a feasible solution of a system of inequalities is found by successively projecting onto violated inequalities. Shor later realized (circa 1970) that his method could be improved by appropriately transforming the space at each iteration (an idea not completely unrelated to Karmarkar's method for linear programming).

The second line of development originates with work by A. Ju. Levin in 1965 in which he discussed a method of "central sections" for general convex programming. D. B. Judin and A. S. Nemirovskii later noticed (1976) that if ellipsoids were used in Levin's method, then it became more efficient and that using these ellipsoids could be viewed as using a particular choice of transformation in Shor's method. In addition, they proved that for a certain class of problems the ellipsoid method could be used to approximate the optimal solution to within a given accuracy σ in time polynomial in the "size" of the problem and $\log 1/\sigma$.[1]

[1] $\log n$ stands for $\log_2 n$

Finally, in 1979 Khachian proved that for LPs with integral data one could get an exact solution in polynomial time. It was this result, brought to the attention of the western mathematical programming community at the 1979 Oberwolfach meeting in Germany, that caused such a stir. It solved the long-standing problem of finding a theoretically efficient algorithm for LPs. The method has not, however, proved effective as a practical method for solving LPs. Its importance for us is based on its theoretical implications in combinatorial optimization.

Our development of the ellipsoid method proceeds as follows. The method is most directly viewed as a method for testing the feasibility of systems of linear inequalities, that is, as a method for testing whether a polyhedron $P = \{x : Ax \leq b\} \subseteq \mathbf{R}^n$ is nonempty. Thus, we begin by showing that testing feasibility is enough to solve LPs. Having made this reduction, we describe the ellipsoid method under two restrictive assumptions: that P is bounded and either empty or ***full dimensional***, $\dim P = n$. These assumptions remove certain technical difficulties, making the presentation more direct. Their relaxation is treated in the exercises.

6.2 Reduction to Testing Feasibility

The first step in the reduction is to apply LP duality theory. Consider the LP

$$\begin{array}{lll} & \min & c^T x \\ (\mathrm{P}) & \text{s.t.} & Ax \geq b \\ & & x \geq 0 \end{array}$$

The dual of (P) is the problem

$$\begin{array}{lll} & \max & b^T y \\ (\mathrm{D}) & \text{s.t.} & A^T y \leq c \\ & & y \geq 0 \end{array}$$

By the LP strong duality theorem, (4.1.2), we know that x^* is an optimal solution of (P) iff x^* is feasible for (P) and there exists a feasible y^* for (D) such that $c^T x^* = b^T y^*$. But a simple calculation, (4.1.1), shows that if x^* and y^* are feasible to (P) and (D), respectively, then $c^T x^* \geq b^T y^*$. We conclude that testing the feasibility of the following linear inequality system is equivalent to solving (P) (and (D)):

$$\begin{array}{rrcr} Ax & & \geq & b \\ & -A^T y & \geq & -c \\ -c^T x & + b^T y & \geq & 0 \\ x & & \geq & 0 \\ & y & \geq & 0 \end{array}$$

We carry the reduction one step further by showing that testing a system for solvability in polynomial time, and finding a solution if one exists, is equivalent to simply recognizing solvable systems in polynomial time. Thus, we show that having, say, a "subroutine" or "oracle" that recognizes solvable systems implies the existence of a method to actually construct solutions.

Consider a system $Ax \leq b$ and suppose we have a subroutine that recognizes solvability. If the system has no solution, there is nothing to do. Assume the contrary. We then perform two reductions (the second reduction actually being an expansion):

Reduction 1. Remove columns from A, and the corresponding variables from x, until any further removals destroy feasibility. Denote the result by $Ax \leq b$, the same as the original.

Reduction 2. Expand the system $Ax \leq b$ by, for each of the inequalities $a^T x \leq \beta$ in the system, adding the reverse inequality $a^T x \geq \beta$ to the system, when doing so preserves feasibility. Again, denote the final result by $Ax \leq b$.

Clearly, both of the above reductions can be carried out with polynomial number of calls to the assumed subroutine: If A is $m \times n$, Reduction 1 requires at most n calls and Reduction 2 at most m calls.

Lemma 6.2.1 *Let $A''x = b''$ be the system of equations corresponding to the system of inequalities added in Reduction 2. Then $A''x = b''$ has a unique solution.*

Note that this lemma *does* imply the desired result. It implies that we can use a subroutine for recognizing solvability to reduce the problem of solving an inequality system to that of solving some equality system. But we know a method to solve equality systems, Gaussian elimination, and this method runs in polynomial time (as was demonstrated in exercise 5.1).

Proof of (6.2.1). First we prove that after Reduction 1, the columns of the matrix A must be linearly independent. Assume not. Let x be a solution to $Ax \leq b$, and let a^j be any dependent column of A. We can "compensate" for $a^j x_j$ by expressing a^j in the form $a^j = A'z$, where A' is A with a^j deleted. In particular, replacing each component x_k of x, other than x_j, by $y_k = x_k + z_k x_j$, we obtain

$$A'y = Ax - a^j x_j + A'z x_j = Ax - a^j x_j + a^j x_j = Ax \leq b,$$

showing that a^j could have been deleted, a contradiction.

Now we prove that $A''x = b''$ has a unique solution. If the rows of A'' span the rows of A this will follow, because then, by the result of the previous paragraph, the columns of A'' must be linearly independent. Suppose that A'' does not span

the row space of A. Then there is a vector c orthogonal to all the rows of A'', but not all the rows of A. Clearly, by adding an appropriate scalar multiple of c to any solution x of $Ax \leq b$, we preserve $A''x = b''$ and can produce equality in some other inequality. This contradicts the maximality of $[A''\ b'']$. □

6.3 Ellipsoids

The ellipsoid method may be viewed as a kind of higher-dimensional binary search in which, instead of halving an interval at each stage, we halve an ellipsoid. In more detail, suppose $P = \{x : Ax \leq b\}$ is a polyhedron and an "ellipsoid" E containing P is given. Then either the "center" c of E is in P, in which case we are done—P has been shown to be nonempty—or c violates one of the inequalities $a^T x \leq \beta$ of $Ax \leq b$. In the latter case we find an ellipsoid E' containing $\{x \in E : a^T x \leq \beta\}$, and show that E' may be chosen so that its volume is less than the volume of E multiplied by a constant factor less than 1, dependent on the dimension n of the ambient space $\mathbf{R}^n$, but independent of P and E. This gives a geometric decrease in the volume of the ellipsoids generated. Finally we give a "polynomial" lower bound on the volume of P, assuming it is nonempty. *If* P is nonempty we therefore find that the center of a containing ellipsoid *must be* in P before the volume of that ellipsoid becomes too small.

For a vector $x \in \mathbf{R}^n$, define $\|x\| = \sqrt{x^T x}$. $\|x\|$ is the *length* or *Euclidean norm* of x. Let A be an $n \times n$ nonsingular matrix and let $c \in \mathbf{R}^n$. Then an ***ellipsoid*** E with *center* c is a set of the form

$$E = \{x : \|A(x - c)\| \leq 1\}.$$

Note that $y = A(x-c)$ iff $x = A^{-1}y+c$. A transformation of the form $T(y) = By+d$, where B is an $n \times n$ matrix and $d \in \mathbf{R}^n$, is called an *affine transformation*; if B is nonsingular, then T is a *nonsingular* affine transformation. It follows that an ellipsoid is the image under a nonsingular affine transformation, $T(y) = A^{-1}y + c$, of the *unit ball* in $\mathbf{R}^n$, $\mathbf{B}^n = \{y : \|y\| \leq 1\}$.

It will be convenient here to introduce an alternative definition of ellipsoid. A *positive definite* matrix D is a matrix of the form $D = A^T A$ for some nonsingular matrix A. We define the ellipsoid $\text{ell}(c, D)$ with center c by

$$\text{ell}(c, D) = \{x : (x - c)^T D^{-1}(x - c) \leq 1\}.$$

The equivalence to the previous definition is immediate from the definition of positive definite matrix.

The crucial result for showing that ellipsoids can be used to treat systems of linear inequalities is a result showing that any "half ellipsoid" is contained in an ellipsoid the "volume" of which is smaller by a suitable constant multiple.

Perhaps a word should be said here about how volume is defined. We need only some very elementary facts. First, it is clear that whatever definition we choose, it should assign volume 1 to the unit cubes [2]. In general, consider the parallelepiped P spanned by a set of vectors $a^1, \ldots, a^n$ in $\mathbf{R}^n$, that is, the convex hull of the set of all vectors achievable as sums of subsets of these vectors—assuming no degeneracies, there will be exactly 2^n such sums, including the origin (which we take to be the result of the empty sum). Now if we replace one of these vectors a^j by αa^j, for a scalar α, then this should multiply the volume of P by $|\alpha|$. On the other hand, if we replace one of the vectors by its sum with some other vector in this list, say replacing a^k by $a^k + a^j$, $j \neq k$, then this should leave the volume of P unchanged. Where A is the matrix with columns $a^1, \ldots, a^n$, it is easy to see that the above conditions imply $\operatorname{vol} P = |\det A|$.

Proposition 6.3.1 *If A is an $n \times n$ matrix and $P \subseteq \mathbf{R}^n$ is "well behaved," then* $\operatorname{vol} A(P) = |\det A| \operatorname{vol} P$. □

Proof. If $\operatorname{aff} P \neq \mathbf{R}^n$, or A is singular, clearly $\operatorname{vol} A(P) = 0$; otherwise, approximate P as a disjoint union of cubes—that this is possible is what we mean by "well behaved"—and apply the conclusion of the previous paragraph. □

We actually use (6.3.1) only for P a very special polyhedron or an ellipsoid. The applications to polyhedra is in the proof of Lemma 6.3.6.

Corollary 6.3.2 $\operatorname{vol} \operatorname{ell}(c, D) = \sqrt{\det D} \operatorname{vol} \mathbf{B}^n$.

Proof. Write $D^{-1} = A^T A$. Then $\operatorname{ell}(c, D) = \{x : \|A(x - c)\| \leq 1\}$. It follows that $\operatorname{ell}(c, D) = A^{-1}(\mathbf{B}^n) + c$. Hence (6.3.1) implies $\operatorname{vol} \operatorname{ell}(c, D) = |\det A^{-1}| \operatorname{vol} \mathbf{B}^n$.

A second proof, not using (6.3.1), runs as follows. Since D is positive definite, it can be written in the form $D = U^T R U$, where U is orthogonal ($U^T U = I$) and R is diagonal. It should be clear that an orthogonal matrix, the application of which simply rotates the space, does not change volume, and that applying a diagonal matrix multiples volume by the absolute value of its determinant. □

Theorem 6.3.3 *Let $E = \operatorname{ell}(c, D)$ be an ellipsoid with center c, and let $a \in \mathbf{R}^n$ be nonzero. Let $H = \{x : a^T x \leq a^T c\}$, and let $E' = \operatorname{ell}(c', D')$ where*

$$c' = c - \frac{1}{n+1} \frac{Da}{\sqrt{a^T D a}},$$

$$D' = \frac{n^2}{n^2 - 1} \left[D - \frac{2}{n+1} \frac{D a a^T D}{a^T D a} \right].$$

[2] We take here as unit cubes any set that has the form $I_1 \times \cdots \times I_n$, where each $I_j \in \{[0,1], (0,1], [0,1), (0,1)\}$ $(i = 1, \ldots, n)$.

Then $H \cap E \subseteq E'$ and

$$\operatorname{vol} E' < e^{-\frac{1}{2(n+1)}} \operatorname{vol} E.$$

Proof. In order to simplify the details of the proof, we apply two nonsingular affine transformations that have the effect of reducing c to 0, E to a unit ball, and H to $\{x : x_1 \leq 0\}$. The success of this approach relies in the end on the fact that when applied to both E and E', these transformations preserve preserve ellipsoids and preserve ratios of volumes, the latter because of (6.3.1).

Let T_1 is the affine transformation $T_1(x) = A(x - c)$ where $D^{-1} = A^T A$ and A is nonsingular. Then T_1 is nonsingular, $T_1(E) = \mathbf{B}^n$ and $T_1(E') = \operatorname{ell}(c_1, D_1)$ where

$$c_1 = -\frac{1}{n+1}\frac{b}{\sqrt{b^T b}},$$

$$D_1 = \frac{n^2}{n^2-1}\left[I - \frac{2}{n+1}\frac{bb^T}{b^T b}\right]$$

and $b = (A^T)^{-1}a$. This is verified by substituting $x = A^{-1}y + c$ in the definition of $\operatorname{ell}(c', D')$. Clearly $T_1(H) = \{x : b^T x \leq 0\}$. Note also that $D_1 = AD'A^T$.

For the second transformation, let U be an orthogonal matrix such that $Ub = \alpha d$, where $d^T = [1\ 0 \ldots 0]$ and $\alpha > 0$. (Such a U can be obtained by applying, say, the Gram-Schmidt process to a basis for $\mathbf{R}^n$, the first vector of which is b.) Define the affine transformation $T_2(x) = Ux$ and define $T = T_2 T_1$. Since T_1 and T_2 are nonsingular, so is T, and it is straightforward to verify that $T(H) = \{x : x_1 \leq 0\}$, $T(E) = \mathbf{B}^n$ and $T(E') = \operatorname{ell}(c_2, D_2)$ where

$$c_2 = -\frac{1}{n+1}d,$$

$$D_2 = \frac{n^2}{n^2-1}\left[I - \frac{2}{n+1}dd^T\right].$$

Note that $D_2 = UAD'A^TU^T$.

Since D_2 is evidently diagonal and has positive entries on the diagonal, it follows that D' is positive definite. Now to see that $\{x \in \mathbf{B}^n : x_1 \leq 0\} = T(E) \cap T(H) \subseteq \operatorname{ell}(c_2, D_2) = T(E')$ involves a straightforward calculation.

To complete the proof we need to estimate $\operatorname{vol} E' / \operatorname{vol} E$. Since, by (6.3.1), T preserves ratios of volumes, we have

$$\frac{\operatorname{vol} E'}{\operatorname{vol} E} = \frac{\operatorname{vol} \operatorname{ell}(c_2, D_2)}{\operatorname{vol} \mathbf{B}^n}.$$

But by (6.3.2),

$$\begin{aligned}\frac{\operatorname{vol} \operatorname{ell}(c_2, D_2)}{\operatorname{vol} \mathbf{B}^n} &= \sqrt{|\det D_2|} \\ &= \frac{n}{n+1}\left[\frac{n^2}{n^2-1}\right]^{\frac{n-1}{2}}.\end{aligned}$$

Using the fact that $e^x > 1 + x$ for $x \neq 0$, we have

$$\begin{aligned} \frac{\operatorname{vol} E'}{\operatorname{vol} E} &< e^{-\frac{1}{n+1}}\left(e^{\frac{1}{n^2-1}}\right)^{\frac{n-1}{2}} \\ &= e^{-\frac{1}{2(n+1)}}, \end{aligned}$$

as required. □

To continue further we must be able to construct a starting ellipsoid, and get a lower bound on the volume of the given polyhedron. In both cases we do this by bounding the complexity of the solutions of systems of linear equations, and this in turn we do by bounding the complexity of determinants. Precise notions of complexity for rational numbers are essential here.

Let $r = p/q \in \mathbf{Q}$ be a rational number where p and q are relatively prime integers. Define

$$\operatorname{size}(r) = 1 + \lceil \log(|p| + 1) \rceil + \lceil \log(|q| + 1) \rceil,$$

where $\lceil p \rceil$ is the least integer greater than or equal to p. Thus, size(r) is a bound on the number of binary digits needed to represent r. Now let $c \in \mathbf{Q}^n$ be an n-vector and $A \in \mathbf{Q}^{mn}$ an $m \times n$ matrix. Define

$$\operatorname{size}(c) = n + \operatorname{size}(c_1) + \ldots + \operatorname{size}(c_n),$$

$$\operatorname{size}(A) = mn + \operatorname{size}(a_{11}) + \ldots + \operatorname{size}(a_{mn}).$$

Proposition 6.3.4 $\operatorname{size}(\det A) < 2\operatorname{size}(A)$.

Proof. As the result is trivial for 1×1 matrices, we may assume A is $n \times n$, $n > 1$. Let $a_{ij} = p_{ij}/q_{ij}$, where p_{ij} and q_{ij} are relatively prime. Let $\sigma = \operatorname{size}(A)$ and $\det A = p/q$, again relatively prime. We have the following inequalities:

$$|\det A| \leq \prod_{i,j}(|p_{ij}| + 1),$$

$$|q| \leq \prod_{i,j} |q_{ij}| < 2^{\sigma-1},$$

$$|p| \leq |\det A|\,|q| \leq \prod_{i,j}(|p_{ij}| + 1)|q_{ij}| < 2^{\sigma-1}.$$

The first inequality is proved by a simple induction using the fact that $\alpha_1 + \ldots + \alpha_k \leq \prod_{j=1}^{k}(|\alpha_j| + 1)$ for any scalars $\alpha_1, \ldots, \alpha_k$. The third line of inequalities uses the first inequality, together with the fact that $n > 1$ to get the last strict inequality. Combining the last two lines proves the proposition. □

The next step is to deal with "vertex complexity."

Proposition 6.3.5 *If $P = \{x : Ax \leq b\} \neq \emptyset$ and $\sigma \geq \text{size}([a^T\ \beta])$ for all rows $[a^T\ \beta]$ of $[A\ b]$, then the complexity of an extreme point of P is at most $4n^2\sigma$.*

Proof. By Proposition 4.1.3, z is an extreme point of P iff z is the unique solution of some subsystem $A'x = b'$ of $Ax = b$. Applying Cramer's rule, we see that each component z_j of z is given by an expression of the form $\det B / \det A'$, where B is A' with the j^{th} column replaced by b. By hypothesis, $\text{size}(B) \leq n\sigma$ and $\text{size}(A') \leq n\sigma$. Hence, $\text{size}(\det B) < 2n\sigma$ and $\text{size}(\det A') < 2n\sigma$. It follows that $\text{size}(z_j) \leq 4n\sigma - 1$, and so $\text{size}(z) \leq n + n(4n\sigma - 1) = 4n^2\sigma$. □

We need one more preliminary result, a method for bounding the volume of a polyhedron from below. To this end we give an exact formula for the volume of one very special polyhedron:

Proposition 6.3.6 *Let $x^1, \ldots, x^{n+1} \in \mathbf{R}^n$. Then*

$$\text{vol conv}\{x^1, \ldots, x^{n+1}\} = \frac{1}{n!} \left| \det \begin{bmatrix} 1 & \cdots & 1 \\ x^1 & & x^{n+1} \end{bmatrix} \right|$$

Proof. Consider $X_n(\alpha) = \text{conv}\{0, \alpha e^1, \ldots, \alpha e^n\}$ where e^j is the j^{th} unit vector in $\mathbf{R}^n$. We claim that $\text{vol}\, X_n(\alpha) = \alpha^n / n!$. This is certainly true for $n = 1$. Assume it is true for a general n. Then by induction,

$$X_{n+1}(\alpha) = \int_0^\alpha \frac{(\alpha - x)^n}{n!}\, dx,$$

which yields

$$\text{vol}\, X_{n+1}(\alpha) = \frac{\alpha^{n+1}}{(n+1)!}.$$

Now let $X = \text{conv}\{0, x^1, \ldots, x^n\} \subseteq \mathbf{R}^n$ and let $A = [x^1 \ldots x^n]$ be an $n \times n$ matrix. Then clearly $X = A(X_n(1))$, and so by Proposition 6.3.1 and the result of the previous paragraph, we have

$$\text{vol}\, X = |\det A|\, \text{vol}\, X_n(1) = \frac{|\det A|}{n!}.$$

Let $y^j = \begin{bmatrix} 1 \\ x^j \end{bmatrix}$. We can now prove the proposition:

$$\begin{aligned} |\det[y^1 \ldots y^{n+1}]| &= |\det[y^1\ y^2 - y^1 \ldots y^{n+1} - y^1]| \\ &= |\det[x^2 - x^1\ \ldots x^{n+1} - x^1]| \\ &= n!\ \text{vol conv}\{0, x^2 - x^1, \ldots, x^{n+1} - x^1\} \\ &= n!\ \text{vol conv}\{x^1, x^2, \ldots, x^{n+1}\}. \quad \square \end{aligned}$$

We can finally state and prove the validity of a restricted version of the ellipsoid method. Note that the statement of the algorithm includes the use of a operator $\langle\cdot\rangle_p$ that truncates its argument to p binary digits beyond the decimal point. We are forced to include some such operator if, for no other reason, because of the presence of a square root in the update formula for c. This unfortunately complicates the proof of validity, and necessitates several technical lemmas. We do not prove them here, referring the reader rather to [10]. In each case they involve estimates of how the eigenvalues of the matrices D vary under the truncation.

Algorithm 6.3.7 The Restricted Ellipsoid Algorithm

Input: An integer σ and a bounded polyhedron $P = \{x : Ax \leq b\} \subseteq \mathbf{R}^n$ specified by a rational matrix $[A\ b] \in \mathbf{Q}^{m n+n}$ such that size$([a^T\ \beta]) \leq \sigma$ and $a \neq 0$ for each row $[a^T\ \beta]$ of $[A\ b]$. We assume that if $P \neq \emptyset$, then P is full dimensional.

Output: The assertion that $P = \emptyset$, or a point $c \in P$.

Comment: Suppose $P \neq \emptyset$. In the unrestricted version of the algorithm, in which the assumption of full dimensionality (and boundedness) is removed, termination does not occur yielding a point $c \in P$, but only with the assertion that $P \neq \emptyset$—see exercise 6.2. A point in P can then be found using the results of §6.2.

begin
  $\nu := 4n^2\sigma$;
  $N := 32n^2\nu$;
  $p := 5N^2$;
  $M := 2^\nu$;
  $D := M^2 I$ where I is an $n \times n$ identity matrix;
  $c := 0 \in \mathbf{R}^n$;
  if $Ac \leq b$ **then** output c and **stop**;
  for $j := 1$ **until** N **do begin**
    let $[a^T\ \beta]$ be a row of $[A\ b]$ such that $a^T c > \beta$;
    $$c := \left\langle c - \frac{1}{n+1}\frac{Da}{\sqrt{a^T Da}} \right\rangle_p ;$$
    $$D := \left\langle \frac{n^2}{n^2-1}\left[D - \frac{2}{n+1}\frac{Daa^T D}{a^T Da}\right] \right\rangle_p ;$$
    comment $\langle\cdot\rangle_p$ truncates to p binary digits beyond the decimal point;
    if $Ac \leq b$ **then** output c and **stop**;
  end
  output the assertion that $P = \emptyset$;
end

Theorem 6.3.8 *The restricted ellipsoid algorithm is correct and runs in time polynomial in the size of the input.*

Proof. In verifying that the algorithm will actually run, that is, that all the steps are well defined, the only nontrivial part is to show that $a^T Da > 0$ at each iteration. But as part of proving Theorem 6.3.3, we proved that if the matrix D given in statement of the theorem is positive definite, then so is D', and given the choice of p in the algorithm, Theorem 13.2 of [10] then implies $\langle D' \rangle_p$ is positive definite[3]. Since the initial D in (6.3.7) is obviously positive definite, it follows that all subsequent D are (the update formula for D being exactly the corresponding formula from (6.3.3)). But D positive definite and $a \neq 0$ evidently implies $a^T Da > 0$.

As the next step in the proof, we observe that if the algorithm terminates with the assertion that $c \in P$, then this is obviously correct. Suppose the algorithm terminates with the assertion that $P = \emptyset$, but in fact $P \neq \emptyset$. We have assumed that P is bounded, and so it is the convex hull of its extreme points. By (6.3.5) these extreme points have size bounded by $\nu = 4n^2\sigma$. Let $E_0 = \text{ell}(0, M^2 I) = M\mathbf{B}^n$, and let $E_j = \text{ell}(c, D)$ for the c and D obtained after the j^{th} application of the **for** loop in (6.3.7) *assuming no truncation is performed.* Clearly $P \subseteq E_0$ since E_0 is convex and all extreme points of P are in E_0—they all have norm at most M.

Now by (6.3.5) and (6.3.6)

$$\text{vol}\, P \geq n^{-n} 2^{-n\nu} \geq 2^{-2n\nu},$$

and since $E_0 = \mathbf{B}^n \subseteq [-1, 1]^n$, we have

$$\text{vol}\, E_0 \leq 2^n M^n \leq 2^n 2^{n\nu} \leq 2^{2n\nu}.$$

Now applying the above inequalities and (6.3.3) yields

$$\text{vol}\, E_N \leq 2^{2n\nu} e^{-\frac{N}{2n+2}} < 2^{2n\nu} 2^{-8n\nu} = 2^{-6n\nu}$$

Were it not for truncation, this would complete the proof since it implies that the volume of E_N is smaller than that of P, while at the same time $P \subseteq E_N$. Indeed $N = 16n^2\nu$ in the statement of the algorithm suffices for this. For the version that includes truncation and for $N = 32n^2\nu$, as stated in the algorithm, Theorems 13.2 and 13.3 of [10] can be used to show that $\text{vol}\, E_N < 2^n M^n e^{-N/8n}$, and that $P \subseteq \text{ell}(c, 4D)$ where $E_N = \text{ell}(c, D)$ and c and D are the final (truncated) c and D produced by the algorithm. But then, using (6.3.2),

$$\text{vol}\, P \leq \text{vol}\, \text{ell}(c, 4D) = 2^n \,\text{vol}\, E_N < e^{-2n\nu},$$

a contradiction. This proves the theorem. □

[3]This proof uses the facts, consequences of the formulas in (6.3.3), that the maximum eigenvalue of D' is at most 4 times the maximum eigenvalue of D and the minimum eigenvalue at least 1/4 times the minimum eigenvalue.

6.4 Equivalence of Optimization and Separation

As currently stated, (6.3.7) requires explicit knowledge of a defining system $Ax \leq b$ for the given polyhedron P. However, even for some very simple CO problems, such as the MST problem (see Theorem 5.3.1), the size of such a system must necessarily be exponential in the number of variables. What is needed is a version of (6.3.7) in which A can be treated in some implicit way. That such a version is possible was observed by Grötschel, Lovász and Schrijver (1981) ["The Ellipsoid Method and its Consequences in Combinatorial Optimization," *Combinatorica* **1** 169–197—corrigendum: **4** (1984) 291–295], Karp and Papadimitriou (1982) ["On linear characterizations of combinatorial optimization problems," *SIAM Journal on Computing* **11** 620-632], and Padberg and Rao (1980) ["The Russian method and integer programming," BGA Working paper, New York University, New York (to appear in *Annals of Operations Research*)].

The following modified "oracle" version of the ellipsoid algorithm has the above property. It is the same as (6.3.7) except for the '*Input*' and two smaller changes necessitated by this change: The two lines in the algorithm where we check $Ac \leq b$ must be replaced by calls to SEP, and the statement 'let $[a^T \ \beta]$ be a row of $[A \ b]$ such that $a^T c > \beta$;' must be correspondingly modified.

Algorithm 6.4.1 Oracle Version of the Restricted Ellipsoid Algorithm

Input: An integer σ and a bounded polyhedron $P \subseteq \mathbf{R}^n$ specified by a "separation" subroutine or "oracle" SEP such that for each $y \in \mathbf{Q}^n$, $SEP(y)$ runs in time polynomial in size(y) and either asserts that $y \in P$, or returns a rational inequality $a^T x \leq \beta \leq a^T y$ valid for P such that size($[a^T \ \beta]$) $\leq \sigma$ and $a \neq 0$. We assume that if $P \neq \emptyset$, then P is full dimensional.

Output: The assertion that $P = \emptyset$, or a point $y \in P$.

begin
  $\nu := 4n^2\sigma$;
  $N := 32n^2\nu$;
  $p := 5N^2$;
  $M := 2^\nu$;
  $D := M^2 I$ where I is an $n \times n$ identity matrix;
  $c := 0 \in \mathbf{R}^n$;
  if $SEP(c)$ asserts that $c \in P$ **then** output c and **stop**;
  for $j := 1$ **until** N **do begin**
    let $[a^T \ \beta]$ be the inequality returned by $SEP(c)$,
    valid for P but violated by c ($a^T c > \beta$);
    $c := \left\langle c - \frac{1}{n+1} \frac{Da}{\sqrt{a^T D a}} \right\rangle_p$;

$$D := \left\langle \frac{n^2}{n^2-1}\left[D - \frac{2}{n+1}\frac{Daa^TD}{a^TDa}\right]\right\rangle_p ;$$

if $SEP(c)$ asserts $c \in P$ **then** output c and **stop**;

end

output the assertion that $P = \emptyset$;

end

It is a straightforward exercise to check that the proof of (6.3.8) remains valid for this modified algorithm [4]. In particular, note that Proposition 6.3.5 can still be applied since this result does not depend on how many rows the defining system has, only on the complexity of these rows. Note also that because of the truncation, size(c) is bounded by a polynomial in σ, and so the calls to $SEP(c)$ run in polynomial time. Finally, observe that (6.4.1) can be used to optimize. The approach given in §2 is no longer valid when a defining system is not explicitly given. One alternative is to replace it by binary search on objective function values. For a, say, minimization problem, after having found one feasible point, we have an upper bound on the optimal value. But we also have a lower bound, because $P \subseteq M\mathbf{B}^n$. Using these bounds, we can get within any desired number q of digits of accuracy using a number of calls to (6.4.1) polynomial in q. Now, by adding an appropriate perturbation to the objective function, a perturbation with polynomailly bounded size, we can also arrange that the optimal solution is unique and that any solution optimal for the new objective is optimal for the original objective. But, since we have a bound on the number of digits required to specify any extreme point of P, by insisting that q is large enough, we can "guess" what that extreme point is. Thus, we can find the exact optimal, the exact optimal to the original problem.

We might summarize the content of the modified oracle version of (6.3.7) as showing that if we can solve "separation" in polynomial time for a polyhedron, then we can solve (linear) optimization for this polyhedron. As our final result in this section we prove a (simplified version of a) converse due to Grötschel, Lovász and Schrijver (see the 1981 paper referenced in the first paragraph of this section). The proof involves a kind of "polarity." Our version of this proof is brief, in the hopes that the main idea is, in this way, not obscured by technical details. A complete treatment seems to encounter such details, in abundance, at every turn.

[4] The reader may ask why we have not stated the ellipsoid algorithm in the above form at the outset, given that the proof works in essentially the same form? The principal reason is the difficulty in dealing with the full-dimensionality assumption in an oracle setting. In the exercises we have indicated a straightforward method when the polyhedron is specified by an explicit linear-inequality system. However, when it is not, then something more complicated must be done. Papadimitriou and Karp resolve the issue by first assuming an appropriate bound on the size of the output of SEP (we have assumed this as well in our modified statement of the algorithm). Grötschel, Lovász and Schriver show, however, that this is not necessary, that only a bound on "vertex complexity" is needed. Indeed, they show, by use of basis reduction techniques due to Lovász, that a modified version of the algorithm will suffice in this case, if we are willing to apply it n times!

Definition 6.4.2 Given a set $P \subseteq \mathbf{R}^n$, we define the *polar* P^* of P by $P^* = \{x : x^T y \leq 1 \forall y \in P\}$.

Note that the polar of a polyhedron is a polyhedron by Theorem 4.1.4.

We state the following result only for polyhedra, although it is valid for any closed convex set in $\mathbf{R}^n$. The proof for general convex sets requires a separating-hyperplane theorem. For polyhedra, these hyperplanes are given in the definition, and the result is easy.

Lemma 6.4.3 *Let $P = \{x : Ax \leq b\}$ for $b \in \mathbf{R}^n$ and A an $m \times n$ matrix. Then $P^{**} = P$ iff $0 \in P$. (P^{**} denotes $(P^*)^*$.)*

Proof. The necessity of the condition $0 \in P$ is clear: The polar of any set contains 0. To prove the converse, first note that $P \subseteq P^{**}$. Indeed, every vector in P^* has inner product at most 1 with every vector in P, which is the same as saying that every vector in P has inner product at most 1 with every vector in P^*. Suppose $P \neq P^{**}$. Then there exists $z \in P^{**} \backslash P$. But $z \notin P$ implies there is a row $[a^T \ \beta]$ of $[A \ b]$ such that $a^T z > \beta$. Note that $\beta \geq 0$, since $0 \in P$. Suppose $\beta > 0$. Then apparently $a' = a/\beta \in P^*$, contradicting the fact that $z \in P^{**}$ since $z^T a' > \beta/\beta = 1$. Hence, $\beta = 0$. Then $a^T z = \delta > 0$. Let $a' = 2a/\delta$. Then $a' \in P^*$, since for $x \in P$, $a^T x \leq \beta = 0$, and so $x^T a' \leq 0 \leq 1$. But $z^T a' = 2\delta/\delta = 2 > 1$, again contradicting the fact that $z \in P^{**}$. □

For a set $X \subseteq \mathbf{R}^n$ define the *interior* of X, $\operatorname{int} X$, to be the set of all $x^0 \in X$ such that for some $\epsilon > 0$, $\|x - x^0\| < \epsilon$ implies $x \in X$.

Lemma 6.4.4 *If $P \subseteq \mathbf{R}^n$ is bounded, full dimensional and $0 \in \operatorname{int} P$, then P^* is bounded and full dimensional.*

Proof. The assumptions of the lemma imply that for some $\epsilon_1, \epsilon_2 > 0$, $\epsilon_1 \mathbf{B}^n \subseteq P \subseteq \epsilon_2 \mathbf{B}^n$, which implies that $1/\epsilon_2 \mathbf{B}^n \subseteq P^* \subseteq 1/\epsilon_1 \mathbf{B}^n$. □

Theorem 6.4.5 (Grötschel, Lovász, Schrijver) *Let $P \subseteq \mathbf{R}^n$ be a bounded polyhedron with a given integer σ bounding the complexity of any extreme point of P. Assume that if $P \neq \emptyset$ then P is full dimensional, and suppose that an oracle OPT is given such that for each $c \in \mathbf{Q}^n$:*

$OPT(c)$ solves the program $\max\{c^T x : x \in P\}$ in time polynomial in σ and $\operatorname{size}(c)$, that is, $OPT(c)$ either declares that $P = \emptyset$ or gives an optimal solution x^*.

Then in time polynomial in σ we can construct a algorithm SEP such that for each $a \in \mathbf{Q}^n$:

> *$SEP(a)$ either declares that $a \in P$, or gives a vector $c \in \mathbf{Q}^n$ and a scalar $\beta \in \mathbf{Q}$ such that $c^T x \leq \beta$ for all $x \in P$ and $c^T a > \beta$; moreover, $SEP(a)$ runs in time polynomial in* size(a) *and σ.*

Proof. If $P = \emptyset$, then SEP is easy to construct. We can test whether this is the case by solving $\max\{0^T x : x \in P\}$ using $OPT(0)$. In the case that $P \neq \emptyset$ we solve $2n$ additional programs $\max\{c^T x : x \in P\}$ where c ranges over the $2n$ unit vectors in $\mathbf{R}^n$ and their negatives. Let $x^1, \ldots, x^{2n}$ be the solutions of these programs and let $x^0 = (\sum_{i=1}^{2n} x^i)/2n$. Then $x^0 \in \text{int}\, P$. Define $Q = (P - x^0)^*$. Note that $0 \in \text{int}(P - x^0)$ so that $Q^* = P - x^0$ and Q is a bounded, full-dimensional polyhedron, by (6.4.3) and (6.4.4); moreover, note that we have a bound on the complexity of the extreme points of Q, because of Proposition 6.3.5 and the fact that the extreme points of defining system of inequalities for Q (with right-hand-side all 1s). This bound is polynomial in σ.

We construct SEP^* that solves separation on Q in polynomial time. This will complete the proof since it will imply, using the ellipsoid algorithm, that we can construct OPT^* for Q, and hence, repeating the argument, SEP for $Q^* = P - x^0$. Obviously, knowing SEP for $P - x^0$ is the same as knowing it for P.

Now let us consider the construction of SEP^* for Q. Let $w \in \mathbf{Q}^n$, and consider the program $\max\{w^T x : x \in P\}$. Let x^* be an optimal solution of $\max\{w^T x : x \in P\}$. There are two cases.

> *Case 1.* Suppose $w^T x^* \leq w^T x^0 + 1$. Then $w^T x \leq w^T x^0 + 1$ for all $x \in P$, that is, $w^T(x - x^0) \leq 1$ for all $x \in P$. Hence, $w \in Q$.
>
> *Case 2.* Suppose $w^T x^* > w^T x^0 + 1$. Then $x^* \in P$ implies that $y^T(x^* - x^0) \leq 1$ for all $y \in Q$, and yet $w^T(x^* - x^0) > 1$. Thus, we have found a hyperplane separating w from Q. □

An important aspect of (6.4.5) is the way in which it makes concrete the connection between finding a polyhedral description and finding an algorithm for a combinatorial problem. The ellipsoid algorithm itself makes a precise statement about how understanding the polyhedron associated with a combinatorial problem can lead to solving it. The above result proves that in a sense these are equivalent problems: If we can find an algorithm, then implicitly we can find a good description of the polyhedron.

Theorem 6.4.5 also has important concrete algorithmic applications. For example, an important problem in combinatorial optimization is that of minimizing a "submodular function." The only known polynomial-time algorithm for doing that uses (6.4.5). An outstanding open problem is to find a direct algorithm.

Exercises

6.1 In the statement of the Restricted Ellipsoid Algorithm, show how to remove the boundedness assumption. This may be done by finding a bounded polyhedron P' such that $P \cap P' = \emptyset$ iff $P = \emptyset$.

6.2 Let $P = \{x : Ax \leq b\}$ where A is an $m \times n$ rational matrix, and let $\delta = \frac{1}{2}n^{-1}2^{-\nu}$ where $\nu = 4n^2 \text{size}([A\ b])$. Let $p^T = [\delta \dots \delta] \in \mathbf{R}^m$. Let $P^\delta = \{x : Ax \leq b + p\}$. Prove that $P^\delta = \emptyset$ iff $P = \emptyset$. Conclude that the full dimensionality assumption on P in the Restricted Ellipsoid Algorithm can be removed.

Hint: Make use of the following statement of the Farkas Lemma, deducible from the Strong Duality Theorem, (4.1.2): Let A be an $m \times n$ matrix and let $b \in \mathbf{R}^m$. Then $Ax \leq b$ has a solution x iff $y^T b \geq 0$ for each vector $y \geq 0$ with $y^T A = 0$. Note that in this statement, at most n components of y need be positive.

Bibliography

[1] Aho, A., J. Hopcroft and U. Ullman (1974), *The Design and Analysis of Computer Algorithms*, Addison-Wesley

[2] Bixby, R. E. (1982), Matroids and operations research, in: *Advanced Techniques in the Practice of Operations Research* (6 tutorials presented at the Semi-Annual joint ORSA/TIMS meeting, Colorado Springs, 1980); H.J. Greenberg, F.H. Murphy and S.H. Shaw, eds.; North-Holland, New York, pp. 333–458.

[3] Bixby, R. E. (1982), *Combinatorial Optimization* (Notes)

[4] Chvátal, V. (1983), *Linear Programming*, Freeman

[5] Grötschel, M. (1985), *Operations Research I: Skriptum zur Vorlesung im SS* (Notes)

[6] Lawler, E. L. (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York, (out of print)

[7] Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, eds. (1985), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, New York

[8] Mehlhorn, K. (1984), *Data Structures and Algorithms 2: Graph Algorithms and $\mathcal{NP}$-Completeness*, EATCS Monographs on Theoretical Computer Science, Springer, Berlin

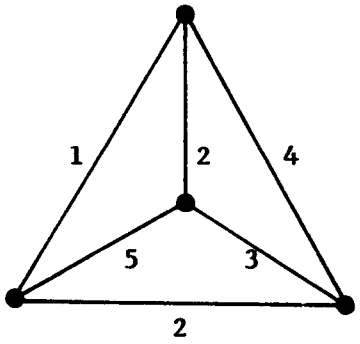
[9] Papadimitriou, C. H. and K. Steiglitz (1982), *Combinatorial Optimization*, Prentice Hall, Englewood Cliffs, New Jersey

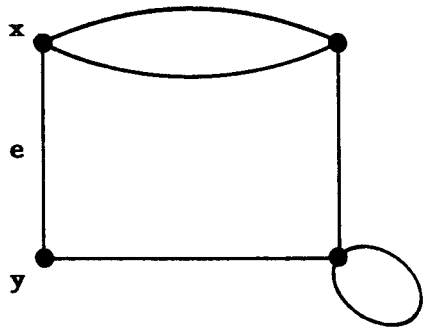
[10] Schrijver, A. (1986), *Theory of Linear and Integer Programming*, Wiley

[11] Tarjan, R. E. (1983), *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania

Notes: [1], [8] and [11] are written by computer scientists. [2] is an introduction to matroid theory, a topic not discussed in any detail in these notes. [4] is an

excellent text on linear programming. [5], written in German, describes an extensive collection of applications, as well as giving an introduction to most of the important topics in combinatorial optimization. [6] and [9] are two standard references on combinatorial optimization. [6] contains a particularly thorough chapter on shortest paths. The emphasis in [9] is on the "primal-dual approach." [10] is an excellent reference work on many aspects of combinatorial optimization, especially, so far as it concerns these notes, on polyhedral combinatorics and the ellipsoid method.


1
2
4
5
3
2


x
e
y